KYMENLAAKSON AMMATTIKORKEAKOULU

Tietotekniikan koulutusohjelma / Ohjelmistotekniikka

Aleksi Grön

MONIALUSTAISEN KÄYTTÖLIITTYMÄN SUUNNITTELUPERIAATTEET

Opinnäytetyö 2014

TIIVISTELMÄ

KYMENLAAKSON AMMATTIKORKEAKOULU
Tietotekniikka

GRÖN, ALEKSI	Monialustaisen käyttöliittymän suunnitteluperiaatteet
Opinnäytetyö	44 sivua + 3 liitesivua
Työn ohjaaja	Laboratorioinsinööri Marko Oras
Toimeksiantaja	Serious Games Finland
Toukokuu 2014	
Avainsanat	käyttöliittymät, käytettävyys, ohjelmointi, cross-platform

Graafiset käyttöliittymät ovat tietokoneiden, mobiililaitteiden ja viihde-elektroniikan kehittymisen ja yleistymisen myötä tulleet tärkeäksi osaksi elämäämme. Yhä useammalla on monia tällaisia laitteita ja yhä useampi haluaa kuluttaa sisältöä monella laitteella. Tämän takia on tärkeä löytää tekniikoita, joilla luoda yhtenäisiä käyttökokemuksia helposti laitteesta riippumatta.

Tässä opinnäytetyössä käytiin läpi mitä vaaditaan hyvältä käyttöliittymältä ja mitä eri osa-alueita niiden suunnittelemiseen kuuluu. Samalla selvitettiin mitä vaihtoehtoja on olemassa monialustaisten käyttöliittymien toteuttamiseen ja mitkä niiden vahvat ja heikot puolet ovat.

Tärkeänä osana työtä suunniteltiin ja toteutettiin käyttöliittymää käyttäen Marmalade SDK:ta kehityksessä olevaan mobiilisovellukseen, jota tullaan käyttämään usealla mobiilikäyttöjärjestelmällä. Iso osa tätä käyttöliittymätoteutusta oli kehittää tekniikoita, joilla voidaan luoda käyttöliittymä, joka toimii mahdollisimman hyvin laitteen näytön koosta riippumatta.

Opinnäytetyön tuloksena syntyi ohjelmistoresursseja, joiden avulla on helppo luoda näytön koosta riippumattomia käyttöliittymiä. Samalla kasattiin paljon tietoa käyttöliittymien suunnittelusta ja käytettävyydestä ja esiin tuli selkeitä jatkokehityskohteita, joihin panostamalla voidaan jatkossa luoda laadukkaita käyttöliittymiä entistä pienemmällä vaivalla entistä useammalle alustalle.

ABSTRACT

KYMENLAAKSON AMMATTIKORKEAKOULU
University of Applied Sciences
Information Technology

GRÖN, ALEKSI	Design principles of cross-platform user interfaces
Bachelor's Thesis	44 pages + 3 pages of appendices
Supervisor	Marko Oras, Laboratory Engineer
Commissioned by	Serious Games Finland
May 2014	
Keywords	user interfaces, usability, programming, cross-platform

Graphical user interfaces have become a big part of our lives as computers and mobile devices have developed and become completely ubiquitous. More people than ever have multiple of these devices and more than ever they want to consume content on many different devices. That's why it is important to find technologies that allow the easy creation of unified user experiences across devices.

One of the main objectives of this thesis was to gather information on what makes a good user interface and what it takes to design one. The goal was also to examine what tools and technologies exist today for creating multiplatform user interfaces and what are the strengths and weaknesses of each one.

A major part of the work was to design and implement a part of the user interface for a mobile application with Marmalade SDK. The application will be run on multiple different mobile operating systems. As a part of the implementation it was necessary to create techniques that allow the user interface to adapt to different screen sizes.

The results of the thesis include software resources that facilitate the relatively easy creation of user interfaces that are independent of screen size. The results also include a large amount of knowledge about user interface design and usability as well as defined targets for future work in this area. Investing in those targets will make the creation of high quality user interfaces for multiple platforms with even less effort.

SISÄLLYS

TERMIT JA LYHENTEET

Alusta: Laitteisto, käyttöjärjestelmä tai niiden yhdistelmä, joka vaikuttaa ohjelmiston toimintaan.

Arkkitehtuuri: Ohjelmistossa suunnitelmallinen korkean tason rakenne.

Metafora: Käyttöliittymän toiminta tai sen osa on vertaus johonkin käyttäjälle tuttuun asiaan fyysisestä maailmasta, antaen lisätietoa käyttöliittymän toiminnasta tai tarkoituksesta.

Kääntää: Muuntaa sovellusprojekti lähdekoodista ja muista resursseista ajettavaksi tiedostoksi.

Kirjasto: Ohjelmoinnissa käytetty kokonaisuus, joka tarjoaa erilaisia toimintoja tai palveluja ohjelmoijalle. Kirjastossa voi olla esimerkiksi luokkia ja funktioita joko lähdekoodi- tai binäärimuodossa.

API (Application Programming Interface): Palvelun, sovelluksen tai kirjaston tarjoama ohjelmointirajapinta.

IwUI: Marmalade SDK:n käyttöliittymäjärjestelmä.

Laskostuminen: Signaalin näytteistäminen muulla kuin alkuperäisellä taajuudella aiheuttaa virheitä. Tämä ongelma esiintyy usein tietokonegrafiikassa käytettäessä tekstuureita.

Bilinear-suodatus: Tekstuurien laskostumisongelmien vähentämiseen käytetty yksinkertainen suodatusmenetelmä.

Mipmap: Laskostumisongelmien vähentämiseksi tekstuurista luodaan useita kokoversioita, joista jokainen on edellistä puolet pienempi. Tekstuuria käytettäessä valitaan sopivin kokoversio ruudulla näytettävän koon perusteella.

Virtuaalinen pikseli: Mittayksikkö, jonka avulla voidaan mitoittaa esimerkiksi käyttöliittymäelementtejä riippumatta käytettävän näytön resoluutiosta.

1 JOHDANTO

Tässä työssä on tarkoitus selvittää käyttöliittymän toteuttamisen ongelmia kun halutaan luoda yhtenäinen käyttökokemus usealla alustalla. Käyttöliittymän toteutuksessa ongelmia tuottaa esimerkiksi syötelaitteiden erilaisuus sekä laitteiden ja käyttöjärjestelmien erot. Käyttäjän pitäisi pystyä käyttämään sovellusta ilman ohjeistusta, käyttäen aikaisemmin sisäistämäänsä tietoa muista käyttöliittymistä.

1.1 Käyttöliittymien tärkeys

Käyttöliittymiä on ympärillämme aivan kaikkialla. Elektroniikan ja muun teknologian kehittymisen myötä digitaalisia käyttöliittymiä on yhä useammassa laitteessa ja niiden tulee mahdollistaa jatkuvasti kasvava määrä toimintoja. Tietokoneissa, televisioissa, pelikonsoleissa, puhelimissa ja tableteissa on yhä suurempi määrä sovelluksia, joiden täytyy luoda helppokäyttöinen käyttöliittymä sovelluksen tarjoamiin ominaisuuksiin.

Jotta voidaan kehittää helppokäyttöisiä sovelluksia, tarvitsee ymmärtää käyttöliittymäsuunnittelun monet eri osa-alueet. Käyttäjä kommunikoi sovelluksen tai palvelun kanssa itse käyttöliittymän lisäksi monilla muilla tavoilla. Kokonaisvaltaiseen palvelun käyttökokemukseen liittyy käyttöliittymän lisäksi paljon asioita, jotka jäävät monesti vähemmälle huomiolle.

Käyttäjän kokemukseen vaikuttaa osana hyvää suunnittelua epäsuorasti myös toteutusteknologian valinta. Toteutusteknologiaa valittaessa tarvitsee määrittää vaadittava laatutaso, jotta osataan valita kustannustehokas, mutta samalla tarpeeksi laadukkaan tuotteen aikaansaava prosessi. Tietyillä teknologioilla on mahdollista saavuttaa parempi laatu lopputuotteessa, mutta useimmiten se tarkoittaa myös suurempaa työmäärää kehitysvaiheessa. Käyttämällä eri teknologioita eri alustoilla voidaan ohittaa useita ongelmia, mutta samalla toteutuskustannukset kasvavat.

Eri teknologiat tarjoavat kehittäjälle mahdollisuuksia ja apua sovelluksen toteuttamiseen. Toiset teknologiat antavat mahdollisuuden tehdä melkein mitä tahansa, mutta kehittäjä joutuu tekemään enemmän työtä itse. Toisilla teknologioilla ei pysty toteuttamaan niin laajaa kirjoa sovelluksia, mutta ne tarjoavat paljon apua tietynlaisiin ongelmiin, vähentäen kehittäjän taakkaa.

1.2 Serious Games Finland

Serious Games Finland on startup-yritys, joka keskittyy hyötypelien ja pelillisten palvelujen toteuttamiseen. Tämä opinnäytetyö tähtää auttamaan Serious Games Finlandin tulevan mobiilisovelluksen kehittämisessä selvittämällä käyttöliittymäsuunnitteluun ja hyvän käyttökokemuksen luomiseen liittyviä ongelmia ja tarjoamalla niihin käytännönläheisiä ratkaisuja. Kyseiseen sovellukseen liittyvälle palvelulle on myös tarkoitus kehittää lähitulevaisuudessa muita sovelluksia useille eri alustoille, jonka takia on erittäin tärkeää ottaa huomioon alusta- ja laiteriippumattomuus jo aikaisessa kehitysvaiheessa.

2 KÄYTTÖLIITTYMÄ

Käyttöliittymä on käyttäjän ja sovelluksen välinen rajapinta, jonka kautta käyttäjä vastaanottaa tietoa ja vaikuttaa sovelluksen toimintaan. Tässä kappaleessa käydään läpi käyttöliittymien kehitys, käyttöliittymien eri osa-alueet ja niihin liittyvät ongelmat, tarkoituksena antaa laaja yleiskuva käyttöliittymien ominaisuuksista ja merkityksestä.

2.1 Käyttöliittymien kehitys

Tietokoneissa on käytetty komentorivikäyttöliittymiä videopäätteiden kehittämisestä lähtien. Kun komentorivikäyttöliittymät tulivat markkinoille, ne tarjosivat ennennäkemätöntä käytännöllisyyttä ja nopeutta verrattuna aikaisempiin vaihtoehtoihin. Komentorivi oli riittävä ratkaisu niin kauan, kun tietokoneita oli lähinnä työpaikoilla ja asiaan paneutuneiden käsissä, koska heillä oli mahdollisuus ja tarve varata aikaa käyttöliittymän opettelemiseen.

1980-luvun alussa julkaistiin ensimmäiset kaupallisesti saatavilla olevat tietokoneet, joista löytyi graafinen käyttöliittymä. Näiden oli tarkoitus saada entistä suurempi määrä ihmisiä käyttämään tietokoneita, mutta myös mahdollistaa graafisten työkalujen käyttäminen.

Uusien käyttäjien käyttökokemus parani merkittävästi komentorivipohjaisista sovelluksista, koska käyttäjille voitiin näyttää mahdolliset toimintovaihtoehdot, ja näin heidän oli paljon helpompi kokeilla sovelluksen eri ominaisuuksia. Graafisia käyttöliittymiä luotaessa oli ensi kertaa mahdollista käyttää visuaalisia metaforia, vertauksia

fyysisestä maailmasta antaen käyttäjille lisävinkkejä eri käyttöliittymäelementtien toiminnasta.

2.1.1 Mobiililaitteet ja kosketusnäytöt

Mobiililaitteita, joihin on tarvinnut suunnitella digitaalisia käyttöliittymiä, on ollut markkinoilla jo 1990-luvulta lähtien. Tuolloin sovelluksia laitteelle on kehittänyt ainoastaan laitteen valmistaja, joten erittäin harvan kehittäjän tarvitsi miettiä, miten mobiililaitteen käyttöliittymän pitäisi toimia. Tuolloin sovellukset myös usein suunniteltiin tarkasti tietylle laitteelle tai muutamalle samantyyppiselle laitteelle. Käyttöliittymät olivat myös erittäin yksinkertaisia verrattuna nykyisiin älypuhelimiin ja niiden lukemattomiin sovelluksiin.

Vasta viimeisen vuosikymmenen aikana tapahtunut räjähdysmäinen älypuhelimien ja niiden mahdollistamien ladattavien sovellusten kasvu on merkittävästi tuonut esiin mobiililaitteiden käyttöliittymiin liittyvät ongelmat. Nykyään käytännössä kaikissa älypuhelimissa käytettävä kosketusnäyttö luo useita ratkaistavia ongelmia. Puhelimen pieni näyttö pakottaa rajaamaan näytettävää tietoa erittäin tarkasti, jotta käyttöliittymä on helppokäyttöinen ja sisältö helposti luettavaa.

2.1.2 Käyttäjäkeskeinen suunnittelu

Samalla kun digitaalisten käyttöliittymien käyttötarkoitus on muuttunut entistä enemmän viihteelliseksi ja vähemmän ammatilliseksi, on myös käyttäjäkunta muuttunut vähemmän tekniseksi. Useimmat käyttäjät eivät hyväksy vaikeasti käytettäviä käyttöliittymiä, koska heidän ei ole pakko käyttää niitä. Tämä on pakottanut käyttöliittymien suunnitteluprosessin sopeutumaan uudenlaisiin vaatimuksiin ja odotuksiin.

Parantaakseen sovelluksen ja käyttöliittymän hyödyllisyyttä ja käytettävyyttä, täytyy huomioida käyttäjä ja hänen tarpeensa. Käyttäjäkeskeinen suunnitteluun keskittyminen on tarpeen, koska kehittäjillä on aivan erilainen kokemus käyttöliittymistä kuin loppukäyttäjillä. Kehittäjän on helppo ajatella, että kuka tahansa osaa käyttää käyttöliittymää, koska hänkin osaa.

Käyttäjät ovat saaneet enemmän kokemuksia hyvistä käyttöliittymistä, koska niitä on nykyään enemmän saatavilla. Aikaisempien kokemuksien seurauksena käyttäjillä on korkeammat odotukset sovelluksien käytettävyydelle, mikä vaatii sovelluskehittäjiä

panostamaan enemmän käyttöliittymään ja käytettävyyden suunnitteluun, jos he aikovat pärjätä markkinoilla.

2.1.3 Korkeatarkkuuksiset näytöt

Entistä korkeampia tarkkuuksia omaavien näyttöjen tuleminen mobiililaitteisiin aiheuttaa lisää työtä sovelluksen kehittäjälle. Jotta käyttöliittymän osat näyttävät yhtä teräviltä kaikkialla käyttöliittymässä ja käyttöliittymä näyttää mahdollisimman hyvältä erilaisilla näytöillä, pitää kehittäjän ottaa huomioon monia eri tilanteita ja tehdä teknisiä ratkaisuja, jotka mahdollistavat käyttöliittymän toimimisen monilla eri laitteilla parhaalla mahdollisella tavalla.

Pitkän aikaa työpöytätietokoneille kehitettävissä sovelluksissa ei tarvinnut huolehtia erilaisista pikselitiheyksistä, koska lähes kaikki näytöt käyttivät samanlaisia tiheyksiä. Normaali vaihteluväli näytöissä oli 70 - 100 pikseliä per tuuma. Käyttäjä itse valitsi mitä resoluutiota hän käytti, ja se vaikutti vain käytössä olevaan työskentelyalueeseen. Mobiililaitteiden näyttöjen kehityksen seurauksena myös työpöytätietokoneisiin on tullut saataville korkeampitarkkuuksisia näyttöjä.

2.2 Käytettävyys

Käytettävyys koostuu helppokäyttöisyydestä sekä tehokkuudesta ja on tärkeä osa sovelluksen käyttökokemusta. Hyvin suunniteltuna käytettävyys mahdollistaa käyttöliittymän nopean oppimisen ja käyttäjien tehtävien helpon suorittamisen. Käytettävyyttä suunniteltaessa tulee ottaa huomioon käyttäjä ja hänen tarpeensa. Käyttöliittymän avulla käyttäjän pitää voida tehdä ne tehtävät, joita hän haluaa, ilman virheitä ja nopeasti. Käytettävyyttä voidaan parantaa parhaiten testaamalla käyttöliittymää oikeilla käyttäjillä ja kehittämällä sitä iteratiivisesti. (2)

Käytettävyyteen kuuluu käyttöliittymän opittavuus, joka kuvaa kuinka hyvin ensikertaa käyttöliittymää käyttävä pystyy oppimaan sen toiminnan ja kuinka nopeasti hän pystyy suorittamaan haluamiaan tehtäviä. Opittavuutta helpottaa looginen rakenne navigaatiossa ja informaatiossa, jonka avulla käyttäjän on helppo löytää haluamansa. (1)

Käyttöliittymä on helpompi oppia jos se kommunikoi sen eri osien toimintaa ja tarkoitusta käyttäjälle. Tämä kommunikaatio voi sisältää esimerkiksi lyhyitä ohjetekstejä siellä missä tarvitaan ja tarkoitusta tai hierarkiaa viestiviä visuaalisia vihjeitä. Opitta-

vuus paranee myös, jos käyttöliittymä hyödyntää käyttäjän aikaisempaa kokemusta soveltamalla käytäntöjä, jotka ovat tuttuja muista sovelluksista ja käyttöliittymistä. Opittavuuteen liittyy vahvasti myös muistettavuus eli se, kuinka helppo käyttäjän on palata käyttämään käyttöliittymää tehokkaasti. (1)

Osa käytettävyyttä on käyttöliittymän tehokkuus, joka kuvaa kokeneen käyttäjän nopeutta suorittaa haluamiaan tehtäviä. Jotta käyttöliittymä on tehokas ja nopea käyttää, tehtävien suorittamiseen vaaditaan mahdollisimman vähän vaiheita, käyttäjän ei tulisi koskaan odottaa turhaan ja käyttäjää tulisi auttaa välttämään virheitä. (1)

Itse sovelluksen käyttöliittymän ulkopuolelta löytyy paljon vuorovaikutuksia, joiden käytettävyyttä pitää miettiä. Esimerkiksi sovelluksen hankkimisen ja palveluun liittymisen pitäisi olla mahdollisimman yksinkertaista, sovelluksen pitäisi kunnioittaa käyttäjän yksityisyyttä ja käyttää sosiaalista mediaa, sähköpostia ja käyttöjärjestelmän ilmoitustoimintoja luottamuksen arvoisesti.

2.3 Käyttökokemus

Käyttökokemukseen sisältyy kaikki sovelluksen tai palvelun osa-alueet, jotka vaikuttavat käyttäjän kokemukseen. Käyttökokemuksen luomisessa huolehditaan enemmän siitä, mitä käyttäjä tuntee, ja vähemmän siitä miten tehokas hän on käyttäessään käyttöliittymää. (3)

Käyttökokemus on nimensä mukaisesti kokemus, ja sen takia paljon henkilökohtaisempi asia kuin käytettävyys tai muut käyttöliittymään liittyvät ominaisuudet. Tämä henkilökohtaisuus on suurin syy miksi käyttökokemus on lähes mahdoton suunnitella luotettavasti. Käyttökokemuksen suunnitteleminen on ennemminkin käyttökokemuksen pitämistä mielessä, kun suunnitellaan sovellusta ja käyttöliittymää samalla yrittäen löytää asioita, joilla käyttäjän voisi yllättää positiivisesti. (3)

Käyttökokemukseen vaikuttaa erittäin merkittävällä tavalla käytettävyys, koska käyttäjän kokemus käyttöliittymän ja itse sovelluksen tai tuotteen miellyttävyydestä, hyödyllisyydestä ja helppokäyttöisyydestä on jatkuva vaikuttaja. Kokemukseen vaikuttaa myös jokainen käyttäjän vuorovaikutus tuotteen, yrityksen tai brändin kanssa, jonka takia käyttökokemusta luotaessa tai kehitettäessä pitää huomioon ottaa koko yrityksen toiminta. (15)

Itse tuotteen ulkopuolisista käyttökokemukseen vaikuttavista käyttäjän vuorovaikutuksista tuotteen tai yrityksen kanssa löytyy useita konkreettisia esimerkkejä, joihin kuuluu, miten käyttäjä on kuullut tuotteesta, mitä kautta hän on hankkinut sovelluksen tai alkanut käyttää palvelua, miten brändi on näkyvillä julkisuudessa ja sosiaalisessa mediassa ja miten tuotetta markkinoidaan. Nämä kaikki vaikuttavat käyttäjän tunteisiin tuotetta ja brändiä kohtaan ja samalla vaikuttavat käyttökokemukseen. (15)

2.4 Syötelaitteet

Laitteella ja sovelluksessa käytettävä syötelaite pitää ottaa huomioon käyttöliittymän suunnittelussa kaikessa, missä käyttäjä voi antaa syötettä, esimerkiksi painaa painikkeita, siirtyä näkymästä toiseen, vierittää näkymää tai syöttää tekstiä. Koko käyttöliittymä on suunniteltava syötelaite mielessä, jotta voidaan tarjota paras mahdollinen käyttökokemus.

2.4.1 Hiiri ja näppäimistö

Hiiri ja näppäimistö ovat perinteisimmät käyttöliittymien syötelaitteista, koska niitä on käytetty jo yli 30 vuoden ajan. Tuona aikana on opittu, mikä toimii ja mikä ei toimi hiirellä ja näppäimistöllä käytettävissä käyttöliittymissä. Hiiri on suhteellisen tarkka osoitinlaite, jonka ansiosta se toimii hyvin tarkkuutta ja tuottavuutta vaativissa käyttötarkoituksissa. Tuota tarkkuutta ja tehokkuutta on myös helppo väärinkäyttää ja luoda käyttöliittymiä, jotka ovat liian monimutkaisia ja vaikeita käyttää.

2.4.2 Kosketusnäyttö

Verrattuna hiireen, kosketusnäyttö on käytännössä epätarkka syötelaite. Teknisesti kosketusnäyttö ei ole erityisen epätarkka, mutta koska sormea pidetään tavallisesti poissa näytön päältä, tarvitsee se siirtää suhteellisen pitkän matkan jokaisen erillisen painalluksen yhteydessä. Lisäksi sormen kohta, joka koskee näyttöön, ei ole tarkka ja yleensä sormen siirtäminen näytön pintaan aiheuttaa samalla myös sivuttaista liikettä.

Mitä pienempi osuttava kohde on, sitä kauemmin siihen osuminen luotettavasti kestää. Tämä pätee sekä hiireen että kosketusnäyttöihin, mutta kosketusnäytöllä tämä on useammin ongelma yllä mainittujen asioiden takia. Välillä käyttöliittymää suunniteltaessa voi joutua tekemään kompromisseja helpon osuttavuuden ja sovelluksen ominaisuuksien välillä. (4)

Hiirtä käyttäessä käyttäjän on helpompi tietää, mitkä käyttöliittymäelementit ovat klikattavissa, koska elementin ulkonäköä voidaan muuttaa, kun hiiri viedään sen päälle. Tässä voi kuitenkin syntyä epäselvyyksiä käyttäjälle, jos kaikki klikattavat elementit eivät käyttäydy odotetulla tavalla. Kosketusnäytöllä tarvitaan kuitenkin jokin muu tapa kertoa käyttäjälle mitkä elementit ovat klikattavia.

Kosketusnäytöllä on kuitenkin monia hyötyjä, kuten luonnolliset eleet vierittämiselle, pienentämiselle ja suurentamiselle. Näyttöjen rajattu koko rohkaisee sovelluskehittäjiä rajoittamaan sovelluksien monimutkaisuutta, jonka seurauksena usein syntyy sovelluksia, jotka ovat paremmin keskittyneitä niiden tärkeimpään osa-alueeseen, ja sen takia helpompia käyttää.

2.4.3 Näppäinohjaus

Näppäinohjaus eroaa merkittävällä tavalla aikaisemmin esitellyistä syötelaitteista, koska jotta voidaan aktivoida käyttöliittymäelementtejä, pitää ensi valita tuo elementti, yleensä jonkin tyyppisellä suuntaohjauksella. Näppäinohjausta käytetään usein esimerkiksi kosketusnäytöttömissä puhelimissa, peliohjaimissa, televisiokaukosäätimissä. Myös käyttöliittymän ohjaamista tietokoneen näppäimistön kanssa voidaan pitää näppäinohjauksen, vaikka se onkin suhteellisen harvinaista.

Edellä kuvattua valitsemista kutsutaan tässä työssä kohdistamiseksi. Kun elementti on valittu, aktivoiminen tehdään siihen tarkoitukseen merkityllä fyysisellä näppäimellä. Käyttäjälle tulee aina kertoa mikä tuo aktivointinäppäin on. Usein näppäinohjaukseen perustuvissa käyttöliittymissä käytetään toista fyysistä näppäintä edelliseen näkymään palaamiseen ja käynnissä olevan toiminnon perumiseen.

Käyttöliittymän kohdistettavat elementit pitää asetella niin, että käyttäjän on helppo ymmärtää miten niiden välillä siirrytään. Jos kohdistettavia elementtejä on paljon, voi olla kannattavaa tarjota käyttäjälle mahdollisuus liikkua useita elementtejä kerrallaan, tai jakaa elementit useaan näkymään ryhmittelemällä niitä.

2.4.4 Puhe- ja liikeohjaus

Puhe- ja liikeohjaus ovat niin sanottuja luonnollisia syötelaitteita, koska ne eivät tarvitse fyysistä kosketusta käyttäjältä. Ne ovat yleistyneet vasta viime vuosina prosessointitehon ja tarvittavien sensorien kehittymisen seurauksena.

Yksinkertaisia puheohjauskäyttöliittymiä on ollut saatavilla jo jonkin aikaa, mutta niiden tarkkuudessa on ollut parantamisen varaa. Puheohjausjärjestelmä koostuu yleensä ääneen tallentamiseen tarvittavasta laitteistosta ja ohjelmistosta, joka osaa muuttaa puhetta äänisignaalista komennoiksi käyttöliittymälle. Vaikka puheohjaustoimintoja on jo integroitu useisiin käyttöjärjestelmiin, on tämän syötelaitteen kehitys vasta alkuvaiheessa.

Liikeohjaus on varsin uusi teknologia. Ensimmäiset laajasti kuluttajakäyttöön levinneet liikeohjausjärjestelmät löytyvät pelikonsoleista. Näille laitteille ei kuitenkaan ole kehitetty niin paljoa sovelluksia, että liikeohjaukseen liittyvät ongelmat ja mahdollisuudet olisivat laajalti tiedossa. Liikeohjauksessa tarvitaan kohdistamista samalla tavalla kuin näppäinohjauksessa.

Liikeohjausjärjestelmien toimintaperiaatteet vaihtelevat enemmän kuin puheohjauksessa, mutta jakaantuvat pääosin kahteen ryhmään. Joko nämä järjestelmät käyttävät mikromekaanisia tai optisia sensoreita tunnistamaan esimerkiksi kädessä pidettävän laitteen liikettä, tai ne käyttävät kamerajärjestelmiä, jotka osaavat kartoittaa kolmiulotteista tilaa ja seurata esimerkiksi käyttäjän käden liikkeitä.

3 KÄYTTÖLIITTYMÄN JA KÄYTETTÄVYYDEN SUUNNITTELU

Tässä kappaleessa halutaan tarjota käytännönläheisempiä ohjeita ja tekniikoita parempaan käyttöliittymän suunnitteluun. Näiden ohjeiden tärkein tehtävä on antaa työkaluja käyttöliittymän suunnitteluun, joita seuraamalla voidaan kehittää käyttöliittymiä käyttäjäkeskeisemmiksi ja helppokäyttöisemmäksi. Näissä ohjeissa halutaan ottaa huomioon eri laitteisiin ja käyttöjärjestelmiin liittyvät erot, jotta käyttöliittymä voidaan suunnitella toimimaan parhaalla mahdollisella tavalla kaikilla alustoilla.

3.1 Käyttäjien ja käyttötapauksien huomioonotto

3.1.1 Käyttäjä

Käyttöliittymän suunnittelu tulisi käytännössä kaikissa tapauksissa aloittaa keskittymällä käyttäjiin ja heidän suorittamiin tehtäviin käyttöliittymässä. Tämä käyttäjäkeskeinen lähtökohta suunnitteluun varmistaa sen, että käyttäjien vaatimukset sovelluksen toiminnallisuudelle täyttyvät ja sovelluksessa todennäköisemmin saavutetaan hyvä käytettävyys. (2) (5)

Aiemmin mainitut käyttäjien suorittamat tehtävät ovat yksinkertaisia toimintoketjuja, joilla on selkeä päämäärä ja tarkoitus. Tehtäviä voivat olla esimerkiksi uuden tapahtuman lisääminen kalenteriin, viestin kirjoittaminen ja lähettäminen, sisällön etsiminen, artikkelin lukeminen tai kuvagallerian selaaminen. Tehtävän suorittamisella on aina jokin tarkoitus käyttäjälle, ja sen takia niiden tulisi olla mahdollisimman helppoja suorittaa. Useammin suoritettavien tai kauemmin kestävien tehtävien suunnitteluun kannattaa panostaa enemmän aikaa. (5)

Jos ollaan suunnittelemassa olemassa olevaa sovellusta uudelleen tai luomassa uutta ratkaisua olemassa olevaan ongelmaan, on tärkeää tarkastella miten nykyiset käyttäjät toimivat. Tätä tietoa hyödyntämällä voidaan luoda ratkaisu, joka on varmasti nykyistä parempi, ja parhaassa tapauksessa ratkaisee kaikki aikaisemmin esiintyneet ongelmat.

Toisaalta jos ollaan luomassa uudenlaista sovellusta, jolla ei ole suoria kilpailijoita eikä selkeää käyttäjäkuntaa, täytyy kehityksen alkuvaiheessa luottaa omaan visioon, ennen kuin sovellus on valmis oikeiden käyttäjien testattavaksi. Jokaisella sovelluksella kuitenkin on kohdeyleisö, olkoon se miten laaja tahansa.

3.1.2 Suunnittelufilosofia

Kun on selvitetty, ketkä ovat sovelluksen käyttäjiä, mitkä heidän tarpeensa ovat sekä missä ja miten he käyttävät sovellusta, on hyvä miettiä, minkälaisella suunnittelufilosofialla sovelluksen ja käyttöliittymän kehityksessä vastaantulevia päätöksiä tehdään ja ongelmia ratkaistaan. Suunnittelufilosofian valinta voi johtua suoraan siitä, minkälainen kohdeyleisö sovelluksella on, mutta valintaan voi vaikuttaa paljon myös brändi ja yrityksen oma toimintafilosofia. (15)

Suunnittelufilosofia voi olla esimerkiksi keskittyä yksinkertaisuuteen ja helppokäyttöisyyteen, jolloin kaikissa päätöksissä halutaan valita se käyttäjille helpompi vaihtoehto. Tällä tavalla todennäköisesti joudutaan tekemään kompromisseja esimerkiksi ohjelman ominaisuuksissa tai esitettävän informaation määrässä. Filosofia voi olla myös keskittyä käytön tehokkuuteen, jolloin sovellukseen todennäköisesti sisältyy enemmän toimintoja, eikä ole niin helppokäyttöinen.

3.1.3 Käyttötapaukset ja -tilanteet

Sovellusta ja sen käyttöliittymää suunniteltaessa on tärkeää ottaa huomioon erilaiset käyttötapaukset ja tilanteet, joissa sovellusta tullaan käyttämään. Mobiililaitteita pidetään koko ajan mukana, kun työpöytäkoneita tai kannettavia käytetään usein esimerkiksi kotona, töissä tai koulussa. Mobiililaitteita käytetään useammin ja lyhyemmissä jaksoissa, koska ne ovat aina mukana ja ne ovat niin helppoja avata. Koko ajan mukana oleminen tarkoittaa, että useasti ei ole saatavilla nopeaa datayhteyttä, kuten langatonta lähiverkkoa. Liikkeellä käytettävien sovellusten pitäisi toimia hyvin myös hitaamman datayhteyden kanssa, jotta käyttäjät saavat suurimman hyödyn missä tahansa he ovatkin. (6)

Työpöytäkäytössä käyttäjillä on usein paremmat yhteydet ja suuremmat näytöt, he ovat harvempien häiriötekijöiden takia paremmin keskittyneitä käyttöön ja heillä on muutenkin paremmin työskentelyyn optimoidut ympäristöt. Näiden ja monien muiden tekijöiden takia he usein käyttävät enemmän aikaa yhteen istuntoon, mutta istuntoja on toisaalta harvemmin.

Eroja käyttäjän tottumuksiin tai käyttäytymiseen voi aiheuttaa myös se, onko kyseessä natiivi sovellus vai selaimen kautta käytettävä web-sovellus. Työpöytäkäytössä käyttäjät ovat tottuneet siihen, että natiivissa sovelluksessa painamalla hiiren oikeaa painiketta näkyviin tulee valikko, joka tarjoaa lisävaihtoehtoja ja -toimintoja. Web-sovelluksessa vastaava painallus yleensä vain näyttää selaimen tavalliset lisätoiminnot sisällölle, jotka eivät liity millään tavalla itse web-sovelluksen toimintaan.

Kosketusnäyttölaitteella kirjoittaminen on yleensä epätarkempaa ja hitaampaa kuin fyysisellä näppäimistöllä. Tämä pätee vielä enemmän jos käyttäjä on liikkeellä eikä esimerkiksi kotisohvalla tukevassa asennossa. Mobiilisovelluksessa on tärkeä yrittää vähentää käyttäjältä vaadittua kirjoittamista ja tarjota vaihtoehdoksi esimerkiksi listoja, joista käyttäjä voi valita haluamansa tai käyttöjärjestelmän omia kosketusnäytölle optimoituja valitsimia milloin mahdollista.

Sovelluksen suunnittelemiseen vaikuttavia käyttötilanteeseen liittyviä tekijöitä on lukemattomia, ja monet niistä hyvin riippuvaisia itse sovelluksesta. Kun pitää eri käyttötilanteet mielessä suunnitellessa käyttöliittymää, pystyy välttämään monia pahoja käytettävyysongelmia.

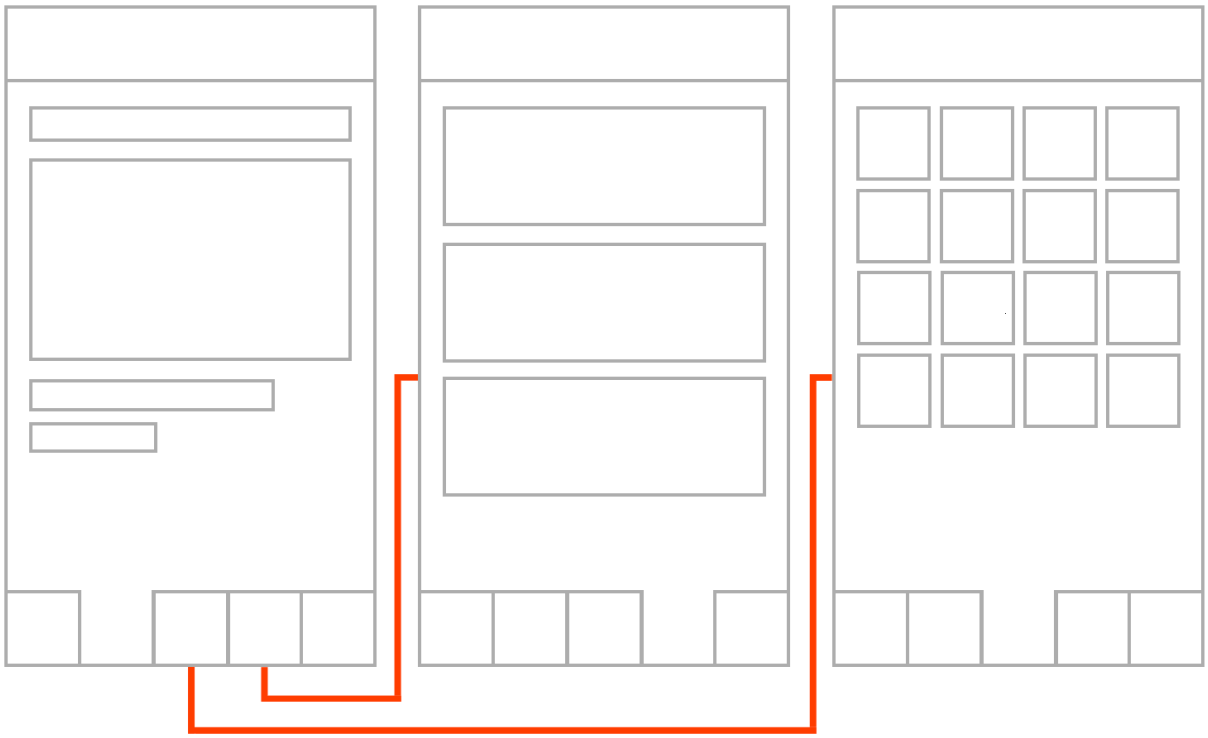
3.2 Navigaatio- ja informaatiosuunnittelu

Navigaation ja informaation rakenteen suunnittelussa päämääränä on jakaa sisältö tarkoituksenmukaiseen rakenteeseen, josta käyttäjän on helppo löytää hakemansa ja ymmärtää näkemänsä tieto.

3.2.1 Navigaatio

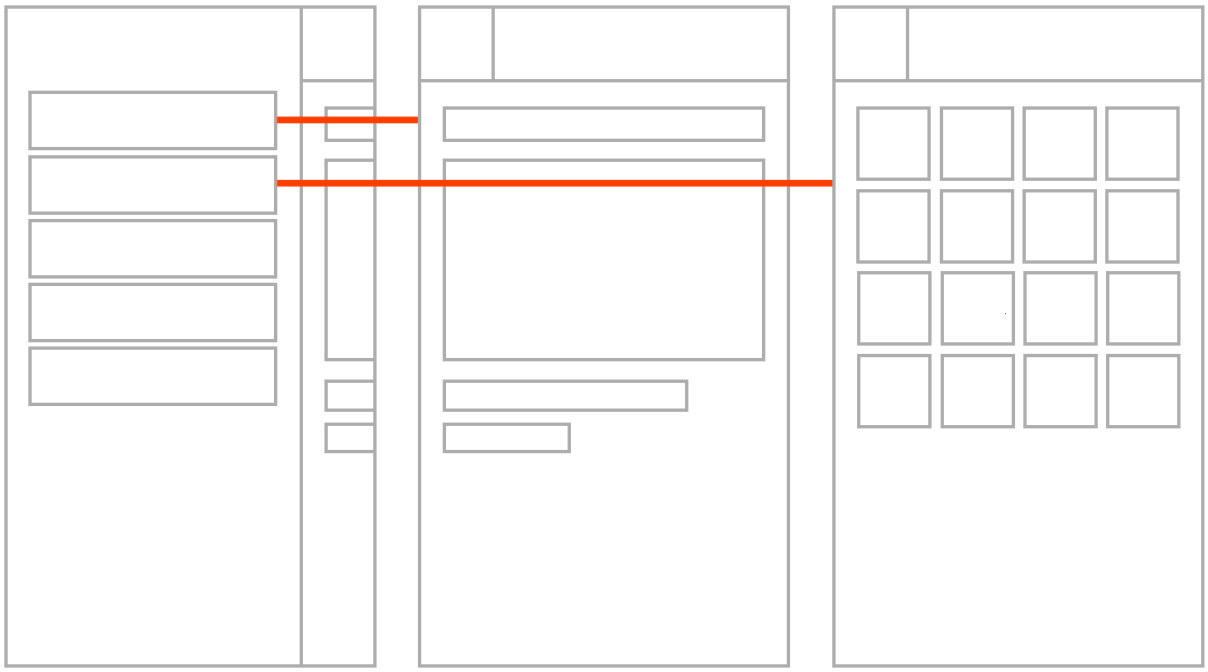
Navigaatio kuvaa millä tavalla käyttäjä liikkuu sovelluksen eri osien ja näkymien välillä ja miten eri näkymät liittyvät toisiinsa. Päätason navigaatioksi kutsutaan ylimmän tason navigaation rakennetta, sitä miten käyttäjät liikkuvat päänäkymien välillä. Tässä kappaleessa olevat esimerkkikuvat ovat puhelimen näytölle suunniteltuja, mutta perusidea on sama myös tableteilla ja työpöytäkoneilla.

Päätason navigaationa voi toimia esimerkiksi välilehdet, joiden välillä käyttäjä voi siirtyä valitsemalla välilehden aina näkyvillä olevasta välilehtipalkista. Välilehtinavigaatio on kuvattu kuvassa 1. Yksittäisten välilehtien tulee tarjota toisistaan erotetut toiminnallisuudet, jotta käyttäjälle on selkeää millä välilehdellä ollaan ja mihin tarkoitukseen kukin välilehti on. Välilehtien käyttämisessä on hyötynä, että käyttäjä voi suoraan siirtyä mille tahansa välilehdelle, mutta koska välilehtipalkki on aina näkyvillä, välilehtien määrää rajoittaa varsinkin mobiililaitteilla näytön koko. Mobiililaitteilla näytön koko rajoittaa myös välilehden kuvaustekstin pituutta, jonka takia käyttäjän voi olla vaikea ymmärtää ja muistaa mihin tarkoitukseen kukin välilehti. Lyhytkin teksti on kuitenkin parempi kuin pelkkä ikoni.



Kuva 1. Välilehtinavigaatio

Jos päänäkymiä on enemmän kuin välilehtipalkkiin mahtuu tai näkymien toiminnallisuutta ei ole selkeästi erotettu toisistaan, voi hyvä vaihtoehto olla laittaa lista näkymistä erilliseen valikkoon. Valikkonavigaatio on kuvattu kuvassa 2. Erillisessä valikossa saadaan paljon vapaammat kädet ulkonäköön ja muotoiluun eikä näytön koko rajaa näkymien määrää, koska valikosta voidaan tehdä vieritettävä. Valikko voi avautua sisältönäkymän päälle, jos käytössä on puhelimen pieni näyttö tai valikko voi näkyä sisällön sivulla tabletilla tai pöytäkoneella. Jos valikko ei ole aina auki, se yleensä avataan sille tarkoitetulla painikkeella.



Kuva 2. Valikkonavigaatio

Usein näkymän sisältöön vaikuttaa aikaisempi valinta, esimerkiksi viestisovelluksessa keskustelun katsominen vaatii ensin keskustelun valitsemista listasta keskusteluita. Keskustelunäkymästä voi lisäksi mennä katsomaan keskusteluun kuuluvien henkilöiden yhteystietosivuja. Tällaista navigaatiorakennetta voidaan kuvata hierarkiana. Navigaatio usein muodostuu hierarkiaksi, koska näkymän sisältö riippuu valinnasta tai näkymästä voi siirtyä useaan eri näkymään.

Käyttöliittymän navigaation on lähes aina yhdistelmä edellä kuvatuista rakenteista. Suunniteltaessa käyttöliittymän navigaatiota on hyvä pitää mielessä, että yksinkertaisempi on yleensä helpompi ymmärtää käyttäjälle. Jos sovelluksen sisältö ja toiminnallisuus jakautuu niin moneen erilliseen osaan, että navigaatiota ei ole helppo ymmärtää, kannattaa miettiä ovatko kaikki toiminnot järkevä pitää sovelluksessa. Toisaalta jos sovelluksen kohdekäyttäjät ovat tehokäyttäjiä, monimutkaisempikin navigaatiorakenne voi toimia. On olemassa myös muita navigaatiorakenteita, mutta yleisimmät niistä ovat muunnelmia tai yhdistelmiä edellä kuvatuista. (6)

3.2.2 Informaatio

Informaatiosuunnittelu liittyy vahvasti navigaatioon, koska sisältö määrää sekä eri päänäkymien suhteet että näkymän sisäisen informaation rakenteen. Informaatiosuun-

nittelu on sisällön jakamista ja ryhmittelemistä erilaisiin kategorioihin ja ryhmiin. Informaatiosuunnittelun tärkeimpänä päämääränä on saattaa käyttäjät heidän haluamansa sisällön pariin mahdollisimman helposti. (7)

Kaikkeen sisältöön liittyy kategorioita, joiden avulla sitä voidaan jakaa ja ryhmitellä. Uutissivustolla sisällön kategorioita voisivat olla esimerkiksi aihepiiri, jonka arvoja olisivat kulttuuri, urheilu ja politiikka, sekä kirjoituksen tyyppi, jonka arvoja olisivat uutinen, haastattelu ja kolumni. Sisältöä voidaan jakaa ensisijaisien ja toissijaisien kategorioiden perusteella. (7)

Ensisijaisia kategorioita ovat ne, jotka kiinnostavat kaikkia palvelun käyttäjiä. Edellä mainitut aihepiiri ja kirjoituksen tyyppi ovat esimerkkejä ensisijaisesta kategoriasta. Toissijaisia kategorioita vastaavasti ovat ne, joiden perusteella vain osa käyttäjistä hakee sisältöä. Esimerkkejä toissijaisista kategorioista ovat vaatteita myyvässä palvelussa tuotteiden merkki, hinta ja arvostelut. Vaikka käyttäjät varmasti ottavat nämäkin asiat huomioon ostopäätöstä tehdessään, suuri osa heistä ei etsi tuotteita niiden perusteella. Kategoriat, joihin sisältö kannattaa jakaa riippuvat paljon sovelluksen tai palvelun kohderyhmästä. (7)

Usein on kannattavaa jakaa päätason navigaatio sisällön ensisijaisten kategorioiden perusteella. Jos ensisijaisten kategorioiden valinnan jälkeen jäljellä on niin paljon sisältöä, että siitä ei helposti löydä etsimäänsä, kannattaa käyttäjille antaa mahdollisuus käyttää toissijaisia kategorioita sisällön rajaamiseen. Toissijaiset kategoriat voivat toimia navigaatiossa hierarkian seuraavana tasona, mutta jos kategorioita yhdistellään usein, voi järkevämpää olla tarjota suodatustoiminnallisuus niiden perusteella. (7)

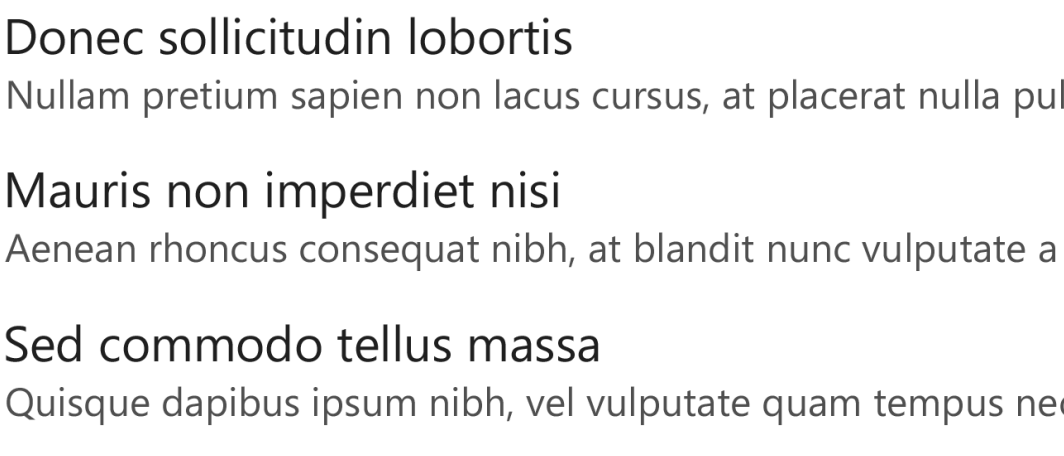
3.3 Graafinen suunnittelu

Käyttöliittymän pitää olla miellyttävä ulkonäöltään, jotta se ei häiritse käyttöliittymän käytettävyyttä tai käyttäjän keskittymistä. Hyvin suunniteltu graafinen ilme voi ilahduttaa ja positiivisesti yllättää käyttäjää, jolloin hän todennäköisemmin palaa käyttämään sovellusta hyvän kokemuksensa takia. Käyttöliittymän graafista suunnittelua pidetään joskus toiminnallisuuden kannalta turhana, mutta sillä voidaan hyvän käyttökokemuksen lisäksi antaa paljon tietoa käyttäjälle käyttöliittymän osien tarkoituksesta ja toiminnasta.

Yhtenäisyys samantyyppisten käyttöliittymäelementtien välillä on erittäin tärkeää käytettävyyden kannalta. Yhtäläisyyksien kautta käyttäjä voi etukäteen tietää mitä kukin elementeistä tekee. Käyttäjä voi esimerkiksi muutaman napin painalluksen jälkeen tietää, että tietyntyyppiset painikkeet avaavat käyttöliittymään uuden näkymän. Samalla tavoin käyttäjä voi oppia, että tietyntyyppiset painikkeet paljastavat lisää tietoa nykyisessä näkymässä. Yhtäläisyydet voivat siirtyä myös sovelluksesta toiseen. Yksi esimerkki on että web-sivuilta tutut siniset, alleviivatut tekstit merkitsevät lähes kaikille käyttäjille linkkejä, jotka vievät toiselle sivulle.

Hiirtä käytettäessä käyttöliittymissä voidaan klikattavia elementtejä muuttaa visuaalisesti kun osoitin viedään niiden päälle merkitsemään käyttäjälle, että elementtiä voi klikata. Kosketusnäytöllä näin ei pystytä tekemään, jolloin käyttäjälle täytyy muin keinoin viestiä mitkä elementit ovat klikattavissa. Klikattaville elementeille käyttää jotain tunnistettavaa tyyliä, jotta käyttäjä ymmärtää klikata niitä. Näitä tyylejä voi olla samassa sovelluksessa useita merkitsemään erilaisia seurauksia napin painamisesta tai kertomaan napin tärkeydestä.

Visuaalisin keinoin voidaan myös ryhmitellä käyttöliittymäelementtejä paremman ymmärrettävyyden saavuttamiseksi. Elementtejä voidaan ryhmitellä lisäämällä ryhmän ympärille reunus tai pitämällä elementit lähellä toisiaan, mutta kauempana muiden ryhmien elementeistä. Ryhmän sisälle voidaan myös luoda visuaalista hierarkiaa esimerkiksi käyttämällä vahvempaa fonttia kohteen otsikolle ja kevyempää fonttia kuvaavalle tekstille. Tämä on esitelty kuvassa 3. (8)



Kuva 3. Visuaalinen ryhmittely

Jos kehitetään sovellusta, joka tulee toimimaan usealla käyttöjärjestelmällä, mutta käyttää samaa käyttöliittymää eri alustoilla, kannattaa käyttöliittymään kehittää oma graafinen ilme, eikä yrittää täysin matkia yhtä tuetuista alustoista. Tällä tavalla käyttöliittymä ei tunnu kompromissilta pääalustan ulkopuolella ja lisäetuna tuotetta ja brändiä voidaan kehittää tunnistettavammaksi. Samalla on kuitenkin tärkeää pitää kiinni tietyistä käyttöjärjestelmien ja muiden sovellusten asettamista käytännöistä, koska niin voidaan välttää monia käytettävyysongelmia.

3.4 Käytettävyyden testaaminen

Paras tapa parantaa sovelluksen käytettävyyttä on testata sovellusta oikeilla käyttäjillä ja aloittaa testaaminen mahdollisimman aikaisin kehitysprosessissa. Monesti sovelluksen kehittäjät saattavat luulla, että siinä ei ole käytettävyysongelmia, koska he eivät itse törmää niihin. Sen lisäksi että kehittäjillä on yleensä vuosien ammatillinen kokemus tietokonejärjestelmien käytöstä, he ovat olleet käyttämässä sovellusta sen koko kehityksen ajan ja ovat samalla oppineet tapoja välttää ja kiertää ongelmia, jotka saattavat saada tavallisen käyttäjän lopettamaan sovelluksen käytön. (9)

4 MONIALUSTAISEN KÄYTTÖLIITTYMÄN TEKNOLOGIAVAIHTOEHDOT

Tässä kappaleessa käydään läpi sovelluksen ja käyttöliittymän toteutusteknologian valitsemiseen liittyviä haasteita ja mahdollisuuksia sekä eri toteutusteknologioiden hyviä ja huonoja puolia. Tärkeitä vaikuttajia teknologiavalinnassa ovat tuotteen toteuttamiseen vaadittu työmäärä ja siihen liittyvä kustannustehokkuus, lopulliselta tuotteelta vaadittava laatutaso ja teknologian käyttöä osaavien työntekijöiden saatavuus.

4.1 Alustojen omat kehitysympäristöt

Käyttöjärjestelmän luoja tarjoaa yleensä sovelluskehittäjille kehitysympäristön mukana käyttöliittymäjärjestelmän, jonka avulla pääsee hyödyntämään kaikkia käyttöjärjestelmän tärkeimpiä ominaisuuksia. Tämäntyyppisten käyttöliittymäjärjestelmien kehittämiseen panostetaan usein erittäin paljon, koska se on käyttöjärjestelmän julkaisuvaiheessa usein ainoa vaihtoehto sovelluskehittäjälle. Nykyaikana käyttöjärjestelmän menestymiselle on ehtona, että sille on saatavilla paljon laadukkaita sovelluksia. Hyvä käyttöliittymäjärjestelmä helpottaa sovelluskehittäjien työtä ja saa näin aikaan enemmän sovelluksia käyttöjärjestelmälle.

Käyttämällä käyttöjärjestelmälle suunniteltua käyttöliittymäjärjestelmää saadaan käyttöön kaikki käyttöjärjestelmän ominaisuudet heti kun ne julkaistaan. Tällä tavalla sovelluksesta saadaan myös visuaalisesti ja käyttäytymiseltään yhtenäinen käyttöjärjestelmän omien sovelluksien kanssa. Käyttäjän on helpompi aloittaa sovelluksen käyttö, kun se on toiminnaltaan hyvin samanlainen hänen aikaisemmin käyttämien sovellusten kanssa.

Toteutettaessa sovellusta käyttöjärjestelmän omilla työkaluilla ohjelmoijien käytössä on useimmiten erittäin laaja kirjasto, josta löytyy ratkaisuja yleisiin ohjelmointiongelmiin. Esimerkkejä tästä ovat kalenteri- ja aikavyöhykeominaisuudet, verkkoyhteyksien hallinta ja datan serialisointi.

Käyttöjärjestelmän omien työkalujen käyttö kuitenkin tarkoittaa, että sovellus täytyy toteuttaa jokaiselle käyttöjärjestelmälle erikseen. Vaikka melko suuri osa sovelluksen suunnittelusta pätee kaikille käyttöjärjestelmille, käytännön työtä joudutaan tekemään silti paljon enemmän. Kuitenkin verrattaessa yhteen usean alustan toteutukseen, alustan omien työkalujen käyttö on todennäköisesti suhteessa nopeampaa.

Kehittäjien täytyy kuitenkin osata käyttää useaa eri järjestelmää, minkä takia lähes pakosti tarvitaan erilliset tiimit tai työntekijät eri alustoille. Esimerkiksi ohjelmointikielien osaamisessa vaaditaan iOS:llä Objective-C:tä, Androidilla Javaa ja Windows Phonella C#:ia. Käytännössä kaikilla alustoilla voidaan käyttää C:tä tai C++:aa, mutta usein työkalut tai muut järjestelmät eivät ole optimoitu siihen.

Vaikka käyttöliittymä ja siihen läheisesti liittyvät järjestelmät toteutettaisiinkin jokaiselle käyttöjärjestelmälle erikseen, voidaan useimmiten uudelleenkäyttää korkean tason logiikkakoodia useammassa toteutuksessa. Tämä tarkoittaa kuitenkin erittäin tarkkaa vastuiden jakoa eri ohjelman osille, jotta sama logiikka toimii kaikilla käyttöjärjestelmillä ja laitetyypeillä. Tällainen järjestelmä on usein vaikea suunnitella, mutta onnistuessaan se tuo mukanaan rakenteeltaan modulaarisen, erittäin hyvin uudelleenkäytettävän koodikannan.

4.2 Marmalade C++ SDK

Marmalade C++ SDK on pelien kehittämiseen keskittynyt järjestelmä, jonka avulla on helppo saada sovellus monelle eri alustalle. Marmalade koostuu laitteistoabstraktion

tarjoavasta Marmalade Systemistä ja korkeamman tason toiminnallisuutta tarjoavasta Marmalade Studiosta. (10)

Marmaladen avulla sovellus on mahdollista kääntää kaikille merkittäville mobiilikäyttöjärjestelmille, Windowsille ja OS X:lle sekä muutamille älytelevisiohin suunnatuille alustoille. Sovelluksen kehittämiseen voidaan käyttää joko Windowsia ja Visual Studiota tai OS X:ää ja Xcodea. Toisin kuin useimmissa muissa kehitysympäristöissä, Marmaladen avulla myös iOS-sovellukset voidaan kääntää ilman Mac-tietokonetta, mikä säästää laitteistokuluissa ja vähentää eri työkalujen opettelutarvetta. (10)

Marmalade tarjoaa helppokäyttöiset projektinhallintatyökalut, joilla saadaan esimerkiksi määriteltyä sovelluksen mukaan pakattavat resurssit, asettaa tietyt alustariippuvaiset sovelluksen ominaisuudet ja siirrettyä sovellus laitteelle testattavaksi.

Koska Marmalade on keskittynyt pelien kehittämiseen, monet tavallisen sovelluksen kehittämiseen tarvittavat ominaisuudet ovat vaillinaisia. Tämän opinnäytetyön kannalta tärkeimpänä esimerkkinä on käyttöliittymäjärjestelmä.

Marmaladen IwUI-käyttöliittymäjärjestelmää ei ole merkittävästi päivitetty noin kolmeen vuoteen (jolloin eri näyttöresoluutiot eivät olleet niin suuri tekijä), mikä tarkoittaa että kehittäjällä ei ole käytettävissä kaikkia työkaluja, joita tarvitaan modernin käyttöliittymän toteuttamiseen. IwUI:ssa ei ole käytettävissä virallisesti tuettua graafista suunnittelutyökalua, vaikka aikaisemmissa versioissa mukana tullutta UIBuilder-sovellusta voikin käyttää.

Marmaladessa tai IwUI:ssa ei ole työkaluja laitteen näytön fyysisen koon tai pikselitiheyden tunnistamiseen, mikä on lähes pakollista nykyisissä mobiililaitteissa, koska näyttöresoluutiot voivat vaihdella esimerkiksi vanhempien iPhone-puhelinten 320 x 480 pikselistä monien Android-puhelinten 1080 x 1920 pikseliin. Pienelle näytölle suunniteltu käyttöliittymä on täysin käyttökelvoton isommilla näytöillä. Vaikka Marmaladen automaattinen käyttöliittymän asettelujärjestelmä osaakin venyttää käyttöliittymäelementit esimerkiksi koko näytön levyiseksi, tekstin koko ei muutu, jolloin se on lukukelvotonta.

Hyvän käyttöliittymäjärjestelmän puutteen lisäksi Marmaladea vaivaa monessa paikassa puutteellinen tai epäselvä dokumentaatio ja kehittäjäyhteisön vähäinen aktiivi-

suus. Monet kehittäjät ovat näistä puutteista huolimatta valmiita valitsemaan Marmaladen hyvän alustatuen ja laitteistoläheisen ohjelmoinnin vuoksi.

4.3 Xamarin

Xamarin on sovelluskehitystyökalu, joka mahdollistaa sovellusten luomisen Android, iOS, OS X ja Windows käyttöjärjestelmille käyttäen alustojen omia käyttöliittymäjärjestelmiä. Xamarin tarjoaa C#-sidokset eri käyttöjärjestelmien ohjelmointirajapintoihin, jolloin jokaisella alustalla voi käyttää käyttöliittymäjärjestelmän lisäksi kaikkia muitakin rajapintoja. (11)

Tämä toteutusvaihtoehto on monella tavalla hyvin samankaltainen kuin kohdassa 5.1 esitelty. Xamarinilla samaa logiikkakoodia voidaan käyttää kaikilla alustoilla, käyttöliittymästä saadaan täysin alustan mukainen ja saadaan pääsy kaikkiin käyttöjärjestelmän ominaisuuksiin. Eroten aikaisemmin esitetystä tavasta, Xamarinilla kehitettäessä voidaan ohjelmointikielenä käyttää kaikilla alustoilla C#:ia, mikä helpottaa osaavien työntekijöiden löytymistä. Tämän lisäksi kaikkia alustoja hallitaan yhtenäisesti, joka nopeuttaa työskentelyä. (11)

Xamarin käyttää Mono-alustaa, joka on avoimen lähdekoodin toteutus Common Language Infrastructure –standardista (CLI), joka on alunperin lähtöisin Microsoftin .NET-alustasta. CLI-sovellukset ajetaan virtuaalikoneessa, mikä vaikuttaa sovelluksen tehokkuuteen jonkin verran, mutta useimmissa tapauksissa tämä ei käytännössä ole ongelma. (12)

4.4 Web-sovellus

Monialustaisen sovelluksen ja käyttöliittymän voi toteuttaa myös web-sovelluksella, jossa on teknologiavaihtoehtona monia hyötyjä. Tässä työssä esitellyistä vaihtoehdoista web-sovellusta pystyy käyttämään ehdottomasti suurimmalla määrällä laitteita. Web-sovelluksen käyttämiseen tarvitaan vain laite, jossa on internetyhteys ja suhteellisen moderni selain riippuen sovelluksen käyttämistä ominaisuuksista.

Käyttöliittymien kannalta ehkä tärkein web-sovelluksien ominaisuus on niiden erittäin monipuolinen käyttöliittymäjärjestelmä. Käyttämällä HTML- ja CSS-merkintäkieliä voidaan web-sovelluksiin rakentaa todella monipuolisia käyttöliittymiä.

Web-sovelluksia kehitettäessä on käytössä paljon helppokäyttöisiä ominaisuuksia, joiden avulla on verrattain nopeaa toteuttaa erilaisia sovelluksen toimintoja. Tämä helppokäyttöisyys tuo mukanaan myös huonoja puolia. Web-sovelluksessa on käytännössä mahdotonta tietää, kuinka paljon muistia sovellus kuluttaa tai paljon laitteessa on muistia käytettävissä. JavaScript-ohjelmointikielen muistinhallinta on täysin automatisoitu, mikä tarkoittaa, että sovelluskehittäjän on erittäin vaikea hallita tuota prosessia. Usein tämä automaatio aiheuttaa viiveitä käyttöliittymässä, kun ajonaikainen ympäristö yrittää vapauttaa muistia, jota ei enää käytetä. Käyttöliittymäjärjestelmä on myös monipuolisuutensa vuoksi jonkin verran hitaampi ja käyttää enemmän muistia kuin yksinkertaisemmat vaihtoehdot, mikä voi muodostua ongelmaksi monimutkaisissa sovelluksissa.

Web-sovelluksessa on toisiin teknologiavaihtoehtoihin verrattuna vaikeampi tai mahdoton käyttää tiettyjä laitteen tai käyttöjärjestelmän ominaisuuksia, kuten kameraa, kosketusnäyttöeleitä, maksuominaisuuksia, kiihtyvyyssensoria tai muita erikoissensoreita. Viime vuosina HTML5-standardiin on kuitenkin lisätty paljon hyödyllisiä ominaisuuksia, kuten laitteen sijainnin raportointi, sovelluksen paikallinen välimuisti, paikallinen tietokanta ja ilmoitusten näyttäminen joillain käyttöjärjestelmillä.

Web-sovellukset voidaan karkeasti jakaa kahteen ryhmään: palvelimella suoritettaviin ja nykyaikaisempiin käyttäjän selaimessa suoritettaviin sovelluksiin. Molemman tyyppiset sovellukset kuitenkin tekevät jonkun verran työtä yhteyden molemmissa päissä. Palvelimella suoritettavassa sovelluksessa käyttäjän halutessa siirtyä toiseen näkymään useimmissa tapauksissa sivu ladataan kokonaan uudestaan palvelimelta. Selaimessa ajettavissa sovelluksissa logiikka on selaimessa JavaScript-koodissa ja sovellus pitää yhteyttä palvelimeen vain kun se tarvitsee jotakin palvelimen resurssia, kuten tietokantaa tai kuvia.

Web-sovelluksissa voidaan nykyään käyttää myös HTML5-standardiin kuuluvaa Offline Web Applications -tekniikkaa, jolla sovellus voidaan saada toimimaan myös ilman internet-yhteyttä. Tällöin sovelluksen tulee ilmoittaa, mitkä resurssit selaimen tulisi pitää välimuistissaan, jotta sovellus toimisi myös ilman internetyhteyttä. Sovellus voi myös tallentaa tietoa paikallisesti ja synkronoida tiedot palvelimen kanssa, kun internetyhteys on taas käytettävissä. (13)

Kun web-sovelluksesta julkaistaan päivitetty versio, sitä ei tarvitse päivittää käyttäjille erikseen. Tämä helpottaa sovelluksen ylläpitoa, koska ei tarvitse varautua tilanteisiin, joissa sovelluksen vanha versio ei ole yhteensopiva esimerkiksi palvelimen nykyisen data-API:n kanssa.

Web-sovelluksen voi luoda ilman sovelluskehystä tai muita kirjastoja, mutta yleensä käytetään jotain sovelluskehystä helpottamaan kehitysprosessia. Sovelluskehyksen tehtävä on auttaa kehittäjää keskittymään luomaan sovelluksen toimintoja, eikä rutiininomaisia sovelluksen osia, jotka ovat samankaltaiset lähes missä tahansa web-sovelluksessa. Esimerkkejä selaimessa ajettavien web-sovelluksien kehittämiseen luoduista sovelluskehyksistä ovat AngularJS, EmberJS, BackboneJS ja KnockoutJS. (14)

4.5 Paikallinen web-tekniikoilla toteutettu sovellus

Paikallinen web-sovellus on käytännössä web-sovellus, joka näkyy käyttöjärjestelmälle tavallisena sovelluksena. Tämä onnistuu sisällyttämällä sovellukseen yksinkertainen selain, jossa itse käyttöliittymää ja logiikkaa ajetaan. Mobiilikäyttöjärjestelmät tarjoavat usein tähän tarkoitukseen järjestelmän oletusselaimen, jotta voidaan säästää sovelluksen koossa. Tämäntyyppisten sovellusten kehittämiseen on useita työkaluja, esimerkkeinä PhoneGap, Web Marmalade ja Adobe AIR. Useat paikallisten web-sovellusten kehittämiseen tarkoitetusta työkaluista tarjoavat mahdollisuuden käyttää käyttöjärjestelmän ohjelmointirajapintoja tai antavat mahdollisuuden ajaa natiivia koodia, jolloin sovelluksen suorituskyvyn kannalta kriittiset osiot voidaan suorittaa tehokkaammin.

Paikalliseen web-tekniikoilla toteutettuun sovellukseen pätee useimmat tavallisen web-sovelluksen hyödyistä ja haitoista, koska sovelluksen ydin ja itse käyttöliittymä on teknisesti samanlainen. Käyttäjälle sovelluksen käynnistäminen ja muu käyttöjärjestelmän puolella tapahtuva interaktio näyttää samalta kuin natiivissa sovelluksessa. Käyttöliittymän viiveet ovat pienempiä kuin tavallisessa web-sovelluksessa, koska käyttöliittymää ja resursseja ei tarvitse ladata verkkoyhteyden yli, mutta internetistä saatava data on silti ladattava kuten missä tahansa muussa sovelluksessa.

4.6 Muita vaihtoehtoja

Monialustaisen sovelluksen kehitykseen on nykyään saatavilla merkittävä määrä erilaisia kehitystyökaluja, joista monet ovat yllättävän samankaltaisia kuin edellä kuva-

tut. Pelien kehittämiseen on kuitenkin saatavilla enemmän vaihtoehtoja kuin tavallisten sovellusten kehittämiseen, koska peleissä tarvitaan usein vähemmän käyttöjärjestelmän ominaisuuksia eikä käyttäjillä ole samoja odotuksia pelien käyttöliittymiltä.

5 CASE STUDY: PHYSIOTOOLS MOBILE

Serious Games Finland kehittää sovellusta mobiililaitteille avustamaan fysioterapiahoidossa. Sovelluksen on tarkoitus muun muassa tarjota käyttäjälle harjoitusohjeet sähköisessä muodossa ja mahdollistaa helpompi tiedonvaihto asiakkaan ja fysioterapeutin tai muun asiantuntijan välillä.

Tämän sovelluksen käyttöliittymän toteuttamisen suurimpia haasteita ovat lukematon määrä tuettavia näyttökokoja ja -resoluutiota sekä eri käyttöjärjestelmien tapoihin ja käyttäjien odotuksiin mukautuminen. Käyttöliittymä pitäisi toteuttaa usealle mobiilikäyttöjärjestelmällä mahdollisimman vähällä erikoistumisella eri alustoille, koska kehittämiseen on käytössä rajalliset resurssit.

5.1 Marmaladen SDK:n käyttäminen

Tämän sovelluksen kehittämiseen käytetään Marmalade SDK:ta. Marmaladen laajan alustatuen mahdollistamana tulevaisuudessa on tarkoitus hyödyntää tämän sovelluksen kehityksessä syntynyttä tietotaitoa ja ohjelmistoresursseja muiden sovelluksien kehityksessä.

Marmalade toimii C- ja C++-ohjelmointikielillä taaten mahdollisimman paljon vaikutettavuutta ohjelman tehokkuuteen ja optimointiin. Projektissa toimivilla ohjelmoijilla on myös paljon kokemusta C++:sta, mikä helpottaa työntekoa. Marmaladen työkaluilla on myös helppo kääntää sovellus usealle käyttöjärjestelmälle kun alkuvalmistelut on tehty.

Marmaladen EDK-laajennusjärjestelmä takaa, että pystymme käyttämään mitä tahansa käyttöjärjestelmän ominaisuutta, vaikka Marmalade ei suoraan tukisikaan sitä. EDK-laajennukset pitää kuitenkin toteuttaa jokaiselle käyttöjärjestelmälle erikseen, minkä takia näihin kannattaa turvautua vain kun on pakko.

Käyttöliittymän toteuttamisen kannalta Marmaladen valinnassa on useita riskejä. Marmalade ei ole päivittänyt tai uudistanut IwUI-käyttöliittymäjärjestelmäänsä mer-

kittävästi pitkään aikaan, mikä voi tarkoittaa, että tulevaisuudessa sen käyttämiseen ei saada apua tai että bugeja ei korjata. Koska sitä ei ole lähiaikoina merkittävästi uudistettu, IwUI:n ominaisuudet ovat jääneet jälkeen monista muista käyttöliittymävaihtoehdoista.

Voi olla, että sovellusta kehitettäessä joudutaan käyttämään ylimääräistä aikaa toteuttaessa ominaisuuksia, joita toisissa käyttöliittymävaihtoehdoissa oli päässyt käyttämään heti. Sellaisen ominaisuuden toteuttamisessa voi mahdollisesti mennä myös niin kauan, että se ei ole kannattavaa liiketoiminnan kannalta, ja annamme näin etua kilpaileville sovelluksille. Marmalade itsessään on keskittynyt pelien kehittämiseen, minkä takia on mahdollista, että tarvittavia ominaisuuksia puuttuu myös muulta kuin käyttöliittymäjärjestelmästä.

5.2 Graafisen suunnittelutyökalun puuttuminen

Käyttöliittymän toteuttaminen lähti liikkeelle tarkastelemalla, mitä työkaluja Marmaladessa ja IwUI:ssa oli tarjolla käyttöliittymän rakentamiseen ja miten ne toimivat. IwUI:n ohjeistusta lukiessa näkee mainintoja UIBuilderista, käyttöliittymien rakentamiseen tarkoitetusta graafisesta suunnittelutyökalusta. Ohjeistuksesta selviää myös, että UIBuilderia on virallisesti tuettu viimeksi Marmaladen versiossa 5.2.

UIBuilderin lähdekoodi on vapaasti saatavilla, joten sitä voitaisiin käyttää vaikka se ei tulekaan itse Marmalade-asennuksen mukana. UIBuilderin käyttämisessä oli kuitenkin useita ongelmia. Käynnistyäkseen UIBuilder vaati, että aktiivinen Marmalade SDK:n versio oli 5.2 tai vanhempi, mikä aiheutti merkittävää työnteon hidastumista, koska aktiivista versiota täytyi vaihtaa esimerkiksi, kun vaihtoi UIBuilderista kehitettävän sovelluksen testaamiseen käytännössä.

UIBuilderin tuottama käyttöliittymämäärittely voitaisiin vaihtoehtoisesti kirjoittaa käsin. Käsin kirjoittamisessa ongelmana kuitenkin on, että formaattia ei ole määritelty tarkasti missään, ja sen toimimaan saaminen on näin monessa kohtaa kokeilun varassa. UIBuilderin käytön ongelmana oli myös sen epäkäytännöllisyys ja epävakaus. Itse UIBuilderin käyttöliittymä oli yksinkertaisesti vaikea ja epätarkka käyttää. Oli myös muita syitä, miksi tämäntyyppisen työkalun käyttö olisi epäoptimaalinen ratkaisu.

Käyttöliittymästä halutaan olio-orientoitunut, joka tarkoittaa, että eri käyttöliittymäkomponentit sisältävät niille kuuluvan toiminnallisuuden ja vastaavat vain niille kuu-

luvista asioista. Tämäntyyppinen rakenne on vaikea saavuttaa, jos käyttöliittymähierarkia rakennetaan suoraan datasta.

5.3 Käyttöliittymähierarkian luominen koodista

Muutamien vaihtoehtojen kokeilemisen jälkeen käyttöliittymäelementit ja -asettelut päädytään luomaan koodista, samalla käyttäen tekstitiedostoista ladattavia tyylejä elementeille koodin siisteyden ylläpitämiseksi.

5.3.1 Käyttöliittymän jakaminen komponentteihin

Käyttöliittymä on hyvä jakaa komponentteihin, jotka osaavat hoitaa tietyn asian, ja näitä komponentteja tulee voida käyttää useissa eri paikoissa. Koska komponentit ovat uudelleenkäytettäviä, käyttöliittymän kehittämiseen kuluu vähemmän aikaa. Käyttöliittymästä löytyy myös vähemmän bugeja, koska samantyyppinen toiminnallisuus toteutetaan vain yhdessä paikassa.

Käyttöliittymän jakaminen komponentteihin tuo käyttöön myös monia olio-ohjelmoinnin etuja. Käyttöliittymäkomponentteja voidaan monesti testata yksikkö- ja integraatiotesteillä käyttöliittymään liittyvien bugien vähentämiseksi.

Komponentit ovat usein helppokäyttöisempiä ohjelmoijan kannalta. Komponentit, jotka koostuvat useista käyttöliittymäelementeistä, voidaan luoda komponenttia käyttävässä koodissa yhdellä koodirivillä. Esimerkiksi komponentti, jonka tehtävä on näyttää tapahtumaan osallistuvat henkilöt kalenterisovelluksessa, voi tarjota ohjelmointirajapinnan, johon tarvitsee antaa vain näiden osallistujien nimet tekstinä, tai mahdollisesti kokonaiset osallistuja-objektit, jolloin komponentti voi itse päättää miten osallistujat tulee esittää.

5.3.2 Sovelluksessa käytetty rakenne

Kehitettävän sovelluksen koko käyttöliittymä on rakennettu erilaisten komponenttien hierarkiana. *ApplicationUI*-komponentti on juurikomponentti, joka sisältää koko käyttöliittymän. Se myös sisältää suoraan kaksi komponenttia, *MainMenu*- ja *MainView*-komponentit, joista *MainMenu* nimensä mukaisesti sisältää päävalikon, ja *MainView* sisältää navigaatiopalkin ja sillä hetkellä näytettävän sisältönäkymän. Jokainen eri sisältönäkymä on oma komponenttinsa, joita voidaan vaihtaa näytettäväksi *MainView*-komponentissa.

Esimerkkinä luomistamme uudelleenkäytettävistä komponenteista on *ButtonItem*. Se on helppokäyttöinen painikekomponentti, joka osaa näyttää sisältöä muutamalla eri tavalla. Se voi näyttää pienen kuvan ja yksi tai kaksi tekstiä missä tahansa yhdistelmässä visuaalisesti yhtenäisellä tavalla monissa paikoissa sovelluksen sisällä.

5.3.3 Mietittäviä asioita komponenttien suunnittelussa

Erilaisten käyttötilanteiden ja vaihtoehtojen tukeminen komponentissa lisää niiden uudelleenkäytettävyyttä, mutta lisää myös komponentin kehittämiseen käytettävää aikaa. Erilaisten vaihtoehtojen tukeminen myös saa myös miettimään, pitäisikö komponentti jakaa useaksi eri komponentiksi, jotka sopisivat eri tilanteisiin. Jos eri vaihtoehdoilla on sama tarkoitus ja sama data, mutta ne eroavat ulkonäössä, kannattaa ne pitää yhtenä komponenttina.

Yksi kriteeri päätettäessä, pitäisikö jokin käyttöliittymäkokonaisuus luoda tavallisesti suoraan elementteinä vai komponenttina, joka sisältää nuo elementit, on, että käytetäänkö kokonaisuutta useassa paikassa. Jos kokonaisuutta käytetään useassa paikassa toisen komponentin sisällä, mutta sitä ei muuteta merkittävästi luomisen jälkeen, kokonaisuuden luomiseen voi riittää oma funktio. Jos kokonaisuutta kuitenkin muokataan merkittävästi tai monimutkaisilla tavoilla, voi olla hyvä luoda siitä komponentti, vaikka sitä ei käytettäisi useasta paikasta.

5.3.4 Tyylit IwUI-järjestelmässä

Marmaladen IwUI-käyttöliittymäjärjestelmässä elementille voidaan asettaa tyyli, joka asettaa arvoja elementin ominaisuuksille. Tyylin data on jaettu kaikkia tyyliä käyttävien elementtien välillä, joka säästää muistia, mutta myös hiukan hidastaa arvojen lukemista. Tyylissä määriteltyjä ominaisuuksia voidaan ylikirjoittaa yksittäisille elementeille, mutta itse tyylejä ei voi muuttaa ajon aikana.

Tyylit määritellään tekstitiedostoissa Marmaladen ITX-formaatissa. Jotta tyylejä voi käyttää käyttöliittymässä, pitää tyylitiedostot ladata Marmaladen resurssijärjestelmään. Tärkein syy tyylien käyttöön on käyttöliittymäkoodin yksinkertaistaminen, koska tyylejä käytettäessä jokaista ominaisuutta ei tarvitse asettaa erikseen jokaiselle elementille.

Tyylejä päätettiin käyttää käyttöliittymässä sellaisten elementtien ominaisuuksien määrittelyyn, joilla on aina samat arvot. Näitä ominaisuuksia ovat esimerkiksi tekstin väri ja asettelu, elementin taustaväri ja kuvaelementtien tekstuuri. Tyylejä käytetään myös elementtien kokojen ja muiden mittojen määrittelyyn, vaikka näytöllä

5.4 Käyttöliittymän suunnitteleminen

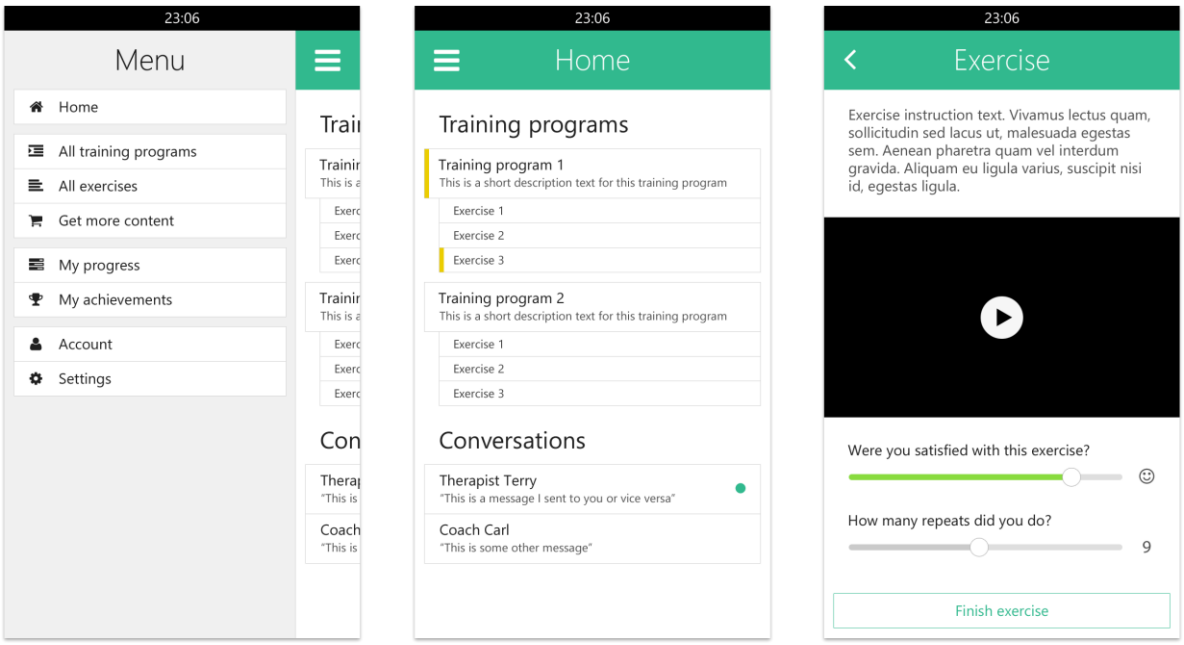
Tässä vaiheessa kehitysprosessia luodaan käyttöliittymää, joka on optimoitu puhelinkokoluokan näytöille, mutta samalla pidetään mielessä, että käyttöliittymää täytyy mukauttaa myöhemmin myös tablet-tietokoneiden näytöille.

Päänavigaatiostrategiana päätettiin käyttää ns. off-canvas-valikkoa, jonka kautta pääsee suoraan siirtymään mihin tahansa päätason näkymään. Päätason näkymiä ovat esimerkiksi kotinäkymä, johon ohjelma tavallisesti aukeaa ja joka pitää sisällään käyttäjälle tärkeimmän tiedon, viestit-näkymä, jossa käyttäjä voi lukea ja lähettää viestejä sekä asetukset-näkymä, jossa käyttäjä voi säätää sovelluksen asetuksia mieleisekseen.

Off-canvas-valikko saa nimensä siitä, että avautuessaan se tulee animaation avustamana esiin näytön ulkopuolelta. Tämäntyyppiset valikot ovat erittäin yleisiä nykyisissä mobiilisovelluksissa ja tuttuja käyttäjille ja ovat sen takia hyvä valinta päänavigaatiostrategiaksi, jos ne sopivat sovelluksen muihin ominaisuuksiin. Navigaatioon liittyvät animaatiot antavat käyttäjälle tietoa käyttöliittymän toiminnasta, minkä ansiosta käyttäjä oppii navigaation toiminnan nopeammin ja ymmärtää paremmin missä osassa sovellusta hän on. Oikealla tavalla hyödynnettynä käyttöliittymäanimaatiot kommunikoivat käyttöliittymän tarkoitusta ja toimintaa, mutta liian pitkät animaatiot tai animaatiot väärässä paikassa voivat aiheuttaa sekaannusta käyttäjässä.

Ennen kuin itse käyttöliittymää alettiin kunnolla toteuttaa Marmaladessa, päätettiin luoda piirroksia käyttöliittymästä, joista osan voi nähdä kuvassa 4. Käyttöliittymäpiirrokset helpottavat ajatusvirheiden huomaamista, käyttöliittymästä keskustelemista ja piirroksia tehdessä ei tarvitse miettiä toteutusta eikä tarvitse tehdä kompromisseja sen takia. Koska virheet voidaan huomata nopeammin, ei tuhlata niin paljon kallisarvoista työtä kuin, jos virhe huomattaisiin vasta kun käyttöliittymä on toteutettu. Sitten kun käyttöliittymää lähdetään oikeasti toteuttamaan, voidaan keskittyä vain tekniseen toteutukseen, eikä tarvitse arvuutella, miltä kunkin käyttöliittymän osan pitäisi näyttää.

Käyttöliittymäpiirroksia on hyvä päivittää jos sovelluksen toiminnallisuuteen tulee suuria muutoksia tai käyttöliittymän rakenne muuttuu.



Kuva 4. Sovelluksen käyttöliittymäpiirroksia

5.5 IwUI-järjestelmän käytön opettelu

Käyttöliittymän suunnittelun jälkeen aloitettiin sen toteuttaminen. Marmaladen IwUI-järjestelmä oli kuitenkin tässä vaiheessa projektia vielä melko uusi tuttavuus, minkä takia sen käyttöä tarvitsi opetella jonkin verran ennen kuin itse käyttöliittymää kannatti lähteä toteuttamaan.

5.5.1 Esimerkkiprojektit

IwUI:n eri ominaisuuksien käytön opettelussa oli erittäin tärkeässä roolissa Marmaladen asennuksen mukana tulevat esimerkkiprojektit, koska muu dokumentaatio oli monessa kohtaa epäselvää tai vaillinaista. Esimerkkejä tutkimalla oli helppo lähteä käyttämään IwUI:n ominaisuuksia omassa käyttöliittymässä. Joissain tilanteissa esimerkkien tutkiminen ei kuitenkaan auttanut, koska ne käyttivät käyttöliittymän luomista tekstitiedostoista.

Päätöksemme luoda käyttöliittymä suoraan koodista aiheutti tässä vaiheessa projektia melkoisen määrän päänvaivaa. Vaikka dokumentaatiossa selvästi sanottiin, että käyt-

töliittymän luominen koodista oli sallittua ja tuettua, ei muussa ohjeistuksessa kiinnitetty juuri yhtään huomiota tähän käyttötapaan. Monet näistä dokumentaation puutteista täytyi korvata yksinkertaisesti kokeilemalla.

5.5.2 Kokeilemalla opittuja asioita

Kun elementtiä lisätään *Layout*-objektiin, elementin asetteluun liittyvät ominaisuudet pitää laittaa *AddElement*-funktiokutsun parametreiksi. IwUI-ohjeistusta lukemalla saa helposti sellaisen käsityksen, että kyseinen funktio lukisi tarvittavat ominaisuudet suoraan elementistä tai sille asetetusta tyylistä, mutta näin ei ole. Tekstitiedostossa määriteltyä käyttöliittymää luotaessa IwUI-järjestelmä kuitenkin tekee tämän automaattisesti, mutta käyttöliittymän koodista luomiseen liittyvä ohjeistus ei mainitse tätä vaatimusta.

Jos halutaan, että *Layout*-objekti ei käytä käyttöliittymässä enempää tilaa kuin on saatavilla, täytyy objektin *sizeToSpace*-ominaisuus asettaa todeksi. Ilman tätä määritystä *Layout*-objekti käyttää niin paljon tilaa kuin sen sisältämät elementit tarvitsevat. Kaiken tarvittavan tilan käyttäminen on haluttavaa silloin, kun on kyse näkymästä jota voi vierittää, mutta useimmissa muissa tapauksissa käyttöliittymäasettelun tulisi sopeutua sille annettuun tilaan.

Vaikka ohjeistuksesta selviää, että kaikilla elementeillä ei ole *margin*-ominaisuutta, on helppo olettaa niin. Tämä ominaisuus kuitenkin löytyy vain *CIwUILabel*- ja *CIwUIButton*-elementeiltä.

IwUI:n hierarkisen tapahtumankäsittelyn vuoksi elementtiä ei voi poistaa hierarkiasta, kun elementti käsittelee tapahtumaa. Tämäntyyppinen tilanne voi tulla vastaan esimerkiksi kun sisältönäkymässä oleva painike aiheuttaa näkymän vaihtamisen toiseen. Näitä tilanteita varten näkymän vaihtaminen pitää myöhästyttää suoritettavaksi vasta tapahtumankäsittelyn jälkeen.

Tähän paras vaihtoehto on *CIwUIElement::UpdateElement*-funktio, jota voidaan käyttää elementtien päivittämiseen. *UpdateElement*-funktiot suoritetaan elementeille *CIwUIView::Update*-funktiossa, joka tapahtuu *CIwUIController::Update*-funktion jälkeen, joka hoitaa tapahtumienkäsittelyn, mutta ennen *CIwUIView::Render*-funktiota, joka piirtää käyttöliittymän ruudulle. Päivitystä haluavien elementtien tarvitsee kutsua

CIwUIElement::SetUpdatable-funktiota, jotta niiden *UpdateElement*-funktiota kutsutaan.

5.6 Riippumattomuus näytön koosta ja resoluutiosta

Laitteet, joilla sovellustamme voidaan käyttää, sisältävät näyttöjä, joilla on hyvin erilaiset resoluutiot. Tämän takia tarvitsemme tekniikoita, joilla pystymme näyttämään käyttöliittymän tavalla, joka on paras mahdollinen kullakin näytöllä. Nämä tekniikat antavat työkaluja, joilla käyttöliittymän osat voidaan mitoittaa sopivan kokoisiksi erikokoisilla näytöillä.

Käytämme mittayksikkönä virtuaalista pikseliä, joka on riippumaton näytön koosta. Voimme käyttää erikokoisilla näytöillä mittayksikköä, joka toimii ennakoitavalla tavalla kaikilla näytöillä. Kaikki käyttöliittymän mitat määritellään virtuaalipikseleinä, ja ne muutetaan käytettävän näytön pikseleiksi kun käyttöliittymähierarkiaa rakennetaan.

Tällä hetkellä virtuaalipikselien muuntaminen näytön pikseleiksi tapahtuu kaavalla $p = vp * (s / vs)$, jossa p on tulos käytettävän näytön pikseleissä, vp on virtuaalipikseliarvo, s on käytettävän näytön lyhyemmän sivun pituus ja vs on virtuaalisen näytön lyhyemmän sivun pituus. Virtuaalinen näyttö kuvaa näyttöä, jonka pikseleillä mittaamme elementtien kokoa. Määrittelyssä käytetään näytön lyhyempää sivua, koska jos käytettäisiin näytön leveyttä tai korkeutta, näytön asento vaikuttaisi käytettävään muunnossuhteeseen, mikä ei ole haluttavaa.

Tällä hetkellä kehityksessä muuttujan *vs* arvo on 720, jolloin esimerkkinä voidaan laskea, että 640 x 960 pikselin näytöllä 20 virtuaalipikseliä olisi 18 näytön pikseliä (20 * (640 / 720)), ja 1080 x 1920 pikselin näytöllä 20 virtuaalipikseliä olisi 30 näytön pikseliä (20 * (1080 / 720)). Voidaan myös ajatella, että yksi virtuaalipikseli on 1 / 720 näytön lyhyemmän sivun pituudesta.

Tällä kaavalla laskettaessa kaikki elementtien koot ovat suhteessa näytön kokoon, jolloin isommalla näytöllä kaikki näyttää isommalta. Useimmissa tilanteissa voisi kuitenkin olla haluttavaa pitää elementit fyysisesti samankokoisina. Ottaakseen tämän huomioon, tarvitsee tietää näytön fyysinen koko.

Marmaladessa ei kuitenkaan ole valmiiksi mitään tapaa saada selville näytön fyysistä kokoa. Tätä varten voitaisiin kirjoittaa laajennus, joka käyttäisi käyttöjärjestelmien omia rajapintoja selvittämään näytön fyysisen koon. Tämä laajennus voitaisiin toteuttaa eri käyttöjärjestelmille Marmaladen EDK-järjestelmällä, joka mahdollistaa käyttöjärjestelmäkohtaisen ohjelmakoodin käyttämisen projektissa.

5.6.1 Elementtien koot ja muut mitat

Virtuaalipikseleissä määritellyt käyttöliittymäelementtien mitat täytyy muuttaa näytön pikseleiksi ennen kuin elementit lisätään käyttöliittymään. Ensimmäinen vaihtoehto oli käyttää tyylimäärityksissä virtuaalipikseleille samoja ominaisuuksia ja tehdä muunnos näyttöpikseleiksi heti, kun tyylitiedostot on ladattu. Tämä ei kuitenkaan ollut mahdollista, koska IwUI-järjestelmä ei anna mahdollisuutta muuttaa tyylimäärityksiä koodista.

Ongelma päädytään ratkaisemaan määrittelemällä mittaominaisuuksille erilliset, virtuaalipikseleissä määriteltävät ominaisuudet. Kun elementtiä ollaan lisäämässä käyttöliittymään, luetaan nämä virtuaalipikseleinä määritellyt ominaisuudet ja asetetaan näytön pikseleiksi muunnetut arvot alkuperäisiin mittaominaisuuksiin. Koska *border*-ominaisuus tarvitsee laittaa parametrina funktioon, joka lisää elementin layout-objektiin, kannattaa tämä heti seuraavaksi.

Koska edellä kuvattu prosessi toistuu todella useasti käyttöliittymähierarkian rakentamisessa, luotiin tähän tarkoitukseen muutama apufunktio. Nämä apufunktiot yksinkertaistavat muuta käyttöliittymäkoodia, koska koko prosessiin riittää useimmiten yksi funktiokutsu. Poikkeuksena tähän ovat elementtityypit, joilla on ylimääräisiä mittaominaisuuksia kuten esimerkiksi *CIwUILabel*- ja *CIwUIButton*-elementit *margin*-ominaisuuksineen. Tällaisille elementeille tulee muistaa kutsua lisäksi funktiota, joka muuntaa niiden ominaisuudet näytön pikseleiksi ennen käyttöliittymään lisäämistä.

Tärkein apufunktioista on *AddElementToLayout*, joka lisää elementin *Layout*-objektiin samalla muuntaen virtuaalipikselimäärityksen näytön pikseleiksi. Pari muuta hyödyllistä funktiota ovat *AddLayoutToLayout*, joka lisää *Layout*-objektin toiseen layout-objektiin ja *AddSpacerToLayout*, joka lisää *Layout*-objektiin välistysobjektin.

Näitä funktioita käyttämällä voidaan käyttää virtuaalipikseleitä myös tekstitiedostossa määriteltävissä tyyleissä, vaikka IwUI ei tätä suoraan tuekaan. Kuitenkin, jotta Mar-

maladen tekstijäsennin ymmärtää, miten lukea käyttämämme ominaisuudet, täytyy sille kertoa, mitä tyyppiä ne ovat. Tämä määritellään Marmaladen resurssijärjestelmän kautta ladattavassa tekstitiedostossa samaan tapaan kuin esimerkiksi tyylit. Määrittely on muotoa *CIwPropertyDefine { nimi tyyppi }*, esimerkiksi *CIwPropertyDefine { marginVP CIwSVec2 }*. Tämäntyyppinen määritys täytyy lisätä jokaiselle itse määrittelemällemme ominaisuudelle.

CIwSVec2 on Marmaladen määrittelemä kaksikomponenttinen vektorityyppi, joka käyttää komponenttien säilömiseen 16-bittistä kokonaislukua. Koska käytämme samaa tyyppiä kuin tavallinen *margin*-ominaisuus, johon laskettu arvo myöhemmin asetetaan, ei arvoa tarvitse muuntaa ennen asettamista.

Edellä kuvattu toteutus on hieman raskaampi kuin kokojen määritteleminen suoraan koodissa johtuen useammista elementtien ominaisuuksien noudoista ja asettamisista, mutta on paljon helpompikäyttöisempi. Emme ole kuitenkaan vielä havainneet huomattavaa vaikutusta sovelluksen tehokkuuteen. Virtuaalipikseleihin liittyvien toimintojen lähdekoodi on liitteessä 1.

5.6.2 Fonttien lataaminen dynaamisesti

Resoluutioriippumattomuuden saavuttamiseksi tarvitaan itse elementtien kokojen määrittelemisen lisäksi fonttien kokojen määritteleminen näytön koosta riippuen. Jos fontit halutaan piirtää optimaalisessa koossa kaikilla mahdollisilla näytöillä, Marmaladen ohjeistuksesta löytyvällä perinteisellä tavalla toteutettuna ohjelman mukaan pitäisi laittaa suuri määrä erikokoisia etukäteen rasteroituja fontteja, joka kasvattaisi ladattavan ohjelman kokoa todella merkittävästi.

Onneksemme Marmaladen *IwGxFont*-moduuli osaa luoda vektoriformaatissa säilöttävistä TTF-fonttitiedostoista piirrettäviä fontteja ajonaikana. Käyttämällä tätä toiminnallisuutta asennettavan ohjelman mukaan laitetaan vain TTF-fonttitiedosto, josta luodaan optimaaliset fonttikoot sovelluksen käynnistyessä kyseisen laitteen näytön mukaan.

Jotta näitä fontteja voidaan käyttää myös tyyleistä, pitää ne lisätä Marmaladen resurssijärjestelmään, mikä on onneksi erittäin yksinkertaista. Ensimmäiseksi luetaan fonttitiedoston sisältö muistiin S3E File -rajapinnan avulla. *IwGxFontCreateTTFontFromBuffer*-funktion avulla luodaan *CIwGxFont*-objekti jokaiselle käyttöliittymässä käytet-

tävälle fonttikoolle ja -variantille. Funktiolle annetaan parametriksi TTF-datan muistialue ja fontin koko, joka lasketaan kohdassa 6.5 kuvatulla tavalla näytön koon mukaan. Kun fontti luodaan *IwGxFontCreateTTFontFromBuffer*-funktiolla itse hallitsemastamme fonttidatasta, täytyy tuon datan olla käytettävissä niin kauan kuin fontteja käytetään.

CIwGxFont-luokka perii *CIwManaged*-luokasta, jonka nimiominaisuutta käytetään resurssin tunnistamiseen tyylimäärityksissä. Tämän takia *CIwGxFont*-objekteille täytyy asettaa nimi, jotta niitä voidaan käyttää tyyleistä. Kun fonttiresursseille on määritetty nimet, voidaan ne lisätä Marmaladen resurssijärjestelmään. Fontteja voidaan käyttää tyylimäärityksissä sen jälkeen, kun fontit on lisätty resurssijärjestelmään. Fonttien lataamiseen ja resurssijärjestelmään lisäämiseen liittyvä lähdekoodi on liitteessä 2.

5.6.3 Käyttöliittymäikonien koot

Käyttämällä aikaisemmin kuvattua tekniikkaa, jossa käyttöliittymässä näkyvien elementtien koot määrätään virtuaalipikseleillä myös käyttöliittymän kuvat ja ikonit voidaan näyttää aina sopivan kokoisena. Näiden kuvien kanssa täytyy kuitenkin päättää, missä koossa niiden kuvadata tallennetaan.

Yksinkertaisin ratkaisu on tallentaa kuvat suurimmassa resoluutiossa jossa niitä tullaan käyttämään, ja skaalata se oikeaan kokoon näytettäessä. Useat kehitystyökalut käyttävät oletuksena skaalaamiseen bilinear-suodatusta, joka toimii hyvin, kun näytettävä koko on lähellä alkuperäistä kokoa, mutta aiheuttaa laskostumisongelmia (*engl. aliasing*) jos kuvaa näytetään paljon pienempänä kuin alkuperäinen versio. Jos näytämme 1080 x 1920 pikselin näytölle optimoituja kuvia 320 x 480 pikselin näytöllä, kokosuhde on vain noin 0,3.

Marmalade osaa onneksi luoda tekstuureista pienemmät mipmap-tasot suhteellisen laadukkaasti, ja IwUI käyttää oikeaa mipmap-tasoa automaattisesti, joten laskostuminen ei muodostu ongelmaksi. Jos sovellus käyttää paljon kuvia, saattaa olla kannattavaa luoda kuvista eri kokoversiot muistinkäytön vähentämiseksi.

Yleensä laitteen keskusmuistin määrä on valittu osittain sen näytön resoluution perusteella, koska isommalla resoluutiolla tarvitaan isompia kuvia, jotta ne näyttävät tarkoilta. Tämä tarkoittaa, että pienemmän resoluution laitteiden muistin määrä ei ole

suunniteltu merkittävästi suurempien kuvien käyttämiseen. Yleensä tämä on ongelma vain sovelluksissa, jotka käyttävät paljon kuvia ja muita resursseja.

Koska nykyisissä mobiililaitteissa näyttöjen koot vaihtelevat todella paljon, kannattaa käyttöliittymäkuvista tehdä useat erikokoiset versiot. Jokaisesta kuvasta tehdään versiot, jotka ovat sopivan kokoisia esimerkiksi 480, 720 ja 1080 pikseliä leveille näytöille. Sovelluksen käynnistyessä päätetään käytössä olevan näytön perusteella mikä versio kuvista ladataan. Erikokoisten kuvien lataamisen helpottamiseksi voidaan käyttää Marmaladen resurssijärjestelmää. Määrittelemällä kuvien erikokoiset versiot omiin resurssiryhmiin voidaan kaikkien kuvien tietty kokoversio ladata yhdellä funktiokutsulla. Nimeämällä kuvien eri kokoversiot samannimisiksi voidaan kuvia käyttää lataamisen jälkeen huolehtimatta siitä, mikä kokoversio on valittu sovelluksen käynnistyksen yhteydessä.

5.6.4 Sisältönä ladattavien kuvien koot

Sovelluksen käyttöliittymässä käytetään sovelluksen mukana tulleiden kuvien lisäksi internetin välityksellä ladattavia kuvia. Näiden kuvien kanssa on käytettävissä hyvin pitkälti samat vaihtoehdot kuin aikaisemmassa kohdassa. Erona tietysti on, että kuvien eri kokoversioita ei voi tulla sovelluksen mukana. Aina ei ole kuitenkaan mahdollista ladata kuvia optimaalisen kokoisena, jolloin ero näytettävään kokoon on vieläkin suurempi kuin käyttöliittymäikonien kohdalla.

Jos sovelluksessa on päädytty käyttämään etukäteen skaalattuja kuvia käyttöliittymän ikoneihin, helpoin vaihtoehto on käyttää oletusskaalaustekniikkaa ladattavissa kuvissa ja antaa kuvanlaadun kärsiä hieman. Toisaalta jos käytettävä kehitystyökalu osaa luoda pienemmät mipmap-tasot ajon aikana laadukkaasti, laskostuminen ei ole ongelma myöskään ladattaville kuville.

5.7 Riippumattomuus syötelaitteesta

Jatkokehityksen kannalta on tärkeää, että nykyisellä prosessilla voidaan tuottaa myös käyttöliittymiä, jotka käyttävät syötelaitteena muuta kuin kosketusnäyttöä. Itse käyttöliittymä pitää todennäköisesti suunnitella erikseen eri syötelaitteille, koska niihin liittyvät käyttötilanteet ovat hyvin erilaisia. Monesti sovelluksen idea ei edes toimi merkittävästi erilaisella alustalla, koska joidenkin asioiden tekeminen on järkevää vain tietynlaisella laitetyypillä.

Nykyinen käyttöliittymien toteutusprosessi sopii kaikille Marmaladen tukemille syötelaitteille, mutta jotain teknisiä ratkaisuja täytyy tehdä, kun aletaan käyttää muita syötelaitteita. Esimerkiksi hiirtä ja näppäimistöä käytettäessä pitää huolehtia, että kaikki tuemme kaikkia tavallisia tapoja navigoida käyttöliittymässä, kuten hiiren rullan ja tiettyjen pikanäppäinyhdistelmien käyttäminen. Näppäinohjausta käytettäessä pitää luoda tapa valita käyttöliittymäelementtejä. IwUI:ssa on onneksi olemassa Focus Handler -toiminnallisuus, joka mahdollistaa kohdistamislogiikan toteuttamisen suhteellisen helposti.

6 YHTEENVETO

6.1 Johtopäätökset

Tämän työn tekemisen aikana on selvinnyt, että Marmaladen ja IwUI:n avulla on mahdollista toteuttaa käyttöliittymä, joka ei riippuvainen laitteesta tai näytön koosta. Riippumattomuus näytön koosta on selkeästi asia, jota IwUI:ta kehittäjät eivät ole osanneet ennakoida kun järjestelmää on kehitetty, koska tuon riippumattomuuden toteuttamiseen ei löytynyt mitään ohjeistusta, eivätkä IwUI:n rajapinnat tarjoa tiettyjä ominaisuuksia, joita tässä olisi kipeästi tarvittu.

Seuraavien sovellusten kehityksen kannalta luomamme käyttöliittymien kehitysprosessi näyttää lupaavalta, koska enää ei tarvitse opetella käyttämään uusia järjestelmiä tai luoda merkittäviä teknisiä ratkaisuja. Joissain asioissa voi vielä parantaa, mutta käyttöliittymien toteuttaminen IwUI:n avulla onnistuu jatkossa varsin mutkattomasti.

Työn aikana on kuitenkin käynyt ilmiselväksi, että Marmalade kokonaisuutena on suunniteltu pelien toteuttamiseen. Yrityksen verkkosivuilla Marmaladea käyttävien julkaistujen tuotteiden listalla on yksi tuote, joka ei ole peli, eikä siinäkään ole kovin

perinteinen käyttöliittymä. Joko kukaan kehittäjä ei ole saanut Marmaladella aikaiseksi tarpeeksi laadukasta tavallista sovellusta tai yritys tällä tavalla vähän kuin tunnustaa muun muassa käyttöliittymäjärjestelmän heikkouden.

Marmaladen tärkein ominaisuus on kuitenkin sen alustatuki, ja sen takia Marmalade valittiin tähän projektiin. On hyvin mahdollista, että jos olisi olemassa vaihtoehto, joka toimisi lähes yhtä monella alustalla ja tarjoaisi paremmat työkalut käyttöliittymien toteuttamiseen, ei olisi päädytty Marmaladeen. Työssä kuvatun sovelluksen kehitys on vielä kesken, mutta käyttöliittymän toteutus on mietitty pääasiassa loppuun asti.

6.2 Työstä opittua

Työn aikana olen ehdottomasti oppinut paljon käyttöliittymien suunnittelusta ja sen osa-alueista, kuten käytettävyydestä, käyttökokemuksesta, käyttäjäkeskeisestä suunnittelusta, käyttötilanteiden huomioonottamisesta, informaatio- ja navigaatiosuunnittelusta, graafisesta suunnittelusta ja käytettävyyden testaamisesta.

Käyttöliittymän suunnittelun ja toteuttamisen ohessa on selvinnyt, kuinka monia asioita pitää ottaa huomioon, kun käyttöliittymää on tarkoitus käyttää usealla eri laitteella, käyttöjärjestelmällä ja näyttökoolla. Monen alustan käyttämisessä on paljon ongelmia, mutta onneksi useimpiin niistä on olemassa hyväksyttävät ratkaisut. Alalla toimivat ammattilaiset oppivat koko ajan uutta näistä ongelmista, luovat niihin uusia ratkaisuja ja parantavat olemassa olevia, joten käyttökokemukset tulevat vain paranemaan tulevaisuudessa.

Käyttöliittymän toteuttamisessa olen joutunut opettelemaan IwUI:n käyttöä aika pitkälti kantapään kautta. Vaikka ohjeistus käy läpi kaikki perusasiat, monet vähänkään edistyneemmät toiminnot ovat dokumentoitu ainoastaan ohjelmointirajapinnan kautta. Lisäksi IwUI:n käyttöön liittyy paljon pieniä sivuseikkoja, jotka pitää tietää ja joita ei mainita ohjeistuksessa tai ne ovat erittäin helppo ymmärtää väärin.

6.3 Kehitettävää jatkossa

Vaikka nykyinen prosessi ja työkalut toimivat suhteellisen hyvin, pitäisi järjestelmää kehittää entistä helpommaksi ja nopeammaksi käyttää. Varsinkin graafisen työkalun puuttuminen on ongelma, jos käyttöliittymää haluaisi toteuttaa ilman syvällistä ohjelmointiosaamista. Ongelmallista tämä on varsinkin siksi, että ohjelmoijat, jotka osaisi-

vat suunnitella käyttöliittymiä, ovat melko harvassa. Toisaalta jos käyttöliittymän suunnitteluun ja toteuttamiseen tarvitaan koko ajan vähintään kaksi eri henkilöä, vaadittu kommunikointi hidastaa työntekoa jonkin verran ja samalla tarvitaan tietysti enemmän työntekijöitä.

Toinen vaihtoehto olisi lähteä kehittämään käyttöliittymän toteuttamiseen järjestelmää, joka ei olisi riippuvainen pohjalla olevasta kehitystyökalusta, jolloin oltaisi vapaat toteuttamaan juuri ne toiminnot, joita tarvitaan, eikä enempää. Vaikka järjestelmää toteutettaisiin vain sen verran kun kussakin projektissa tarvitaan, olisi tämä silti valtava projekti, jonka takia se ei ainakaan lyhyellä aikavälillä ole järkevä vaihtoehto.

Oman järjestelmän luomisen sijaan voisi olla järkevää käyttää jotain olemassa olevaa käyttöliittymäjärjestelmää. Näissä on kuitenkin hyvin usein ongelmana, että ne ovat sidottuja johonkin alustaan tai muutamaan alustaan. HTML5:n, CSS:n ja JavaScriptin yhdistelmä, jota käytetään web-sovellusten toteuttamisessa, olisi varsin varteenotettava vaihtoehto, jos sitä pystyttäisiin käyttämään lähellekään yhtä tehokkaasti kuin natiiveja vaihtoehtoja. Web-tekniikat toimivat vallan mainiosti esimerkiksi työpöytätietokoneissa, koska suoritustehoa on niin paljon, mutta mobiililaitteissa ja todellista nopeutta ja tehokkuutta vaativassa käytössä ei niinkään.

Monen alustan käyttöliittymät ovat niin merkittävä asia, että uskon lähitulevaisuudessa kehitettävän uusia ratkaisuja, jotka poistavat tai vähentävät nykyään niin vahvoja ongelmia, joita tässäkin työssä on yritetty ratkoa.

LÄHTEET

1. Usability 101: Introduction to Usability. 2012. Nielsen Norman Group. Saatavissa: http://www.nngroup.com/articles/usability-101-introduction-to-usability/ [Viitattu 13.3.2014].

2. Designing for usability. 1985. John D. Gould and Clayton Lewis. Saatavissa: http://xa.yimg.com/kq/groups/23186582/23590533/name/UCD_gould.pdf [Viitattu 13.3.2014].

3. Why user experience cannot be designed. 2011. Smashing Magazine. Saatavissa: http://uxdesign.smashingmagazine.com/2011/03/15/why-user-experience-cannot-be-designed/ [Viitattu 28.3.2014].

4. Finger-friendly design: ideal mobile touchscreen target sizes. 2012. Smashing Magazine. Saatavissa: http://www.smashingmagazine.com/2012/02/21/finger-friendly-design-ideal-mobile-touchscreen-target-sizes/ [Viitattu: 30.4.3014].

5. Context of use within usability activities. 2001. Martin Maguire. Saatavissa: http://www.researchgate.net/profile/Martin_Maguire2/publication/222298784_Context_of_Use_within_usability_activities/file/504635217421b75460.pdf [Viitattu 3.4.2014].

6. Designing for Mobile, Part 1: Information Architecture. 2012. UX Booth. Saatavissa: http://www.uxbooth.com/articles/designing-for-mobile-part-1-information-architecture/ [Viitattu 20.3.2014].

7. Efficiently Simplifying Navigation, Part 1: Information Architecture. 2013. Smashing Magazine. Saatavissa: http://uxdesign.smashingmagazine.com/2013/12/03/efficiently-simplifying-navigation-information-architecture/ [Viitattu 20.3.2014].

8. 10 Useful techniques to improve your user interface designs. 2008. Smashing Magazine. Saatavissa: http://www.smashingmagazine.com/2008/12/15/10-useful-techniques-to-improve-your-user-interface-designs/ [Viitattu 28.4.2014].

9. Pragmatic UX Techniques For Smarter Websites. 2014. Smashing Magazine. Saatavissa: http://uxdesign.smashingmagazine.com/2014/01/27/pragmatic-ux-techniques-for-smarter-websites/ [Viitattu 13.3.2014].

10. Marmalade Platform Support. 2014. Marmalade. Saatavissa: https://www.madewithmarmalade.com/marmalade/supported-platforms [Viitattu 13.4.2014].

11. Xamarin Tour. 2014. Xamarin. Saatavissa: http://xamarin.com/tour [Viitattu 14.4.2014].

12. Xamarin: Introduction to mobile development. 2014. Xamarin. http://docs.xamarin.com/guides/cross-platform/getting_started/introduction_to_mobile_development/ [Viitattu 15.4.2014].

13. Let's take this offline. 2011. Dive into HTML5. Saatavissa: http://diveintohtml5.info/offline.html [Viitattu 15.4.2014].

14. Using Frameworks to Build Websites and Web Applications. 2013. OSTraining. http://www.ostraining.com/blog/webdesign/frameworks/ [Viitattu 15.4.2014].

15. A simplified model of user experience for practical application. 2006. Hans-Christian Jetter, Jens Gerken. Saatavissa: http://www.inf.uni-konstanz.de/gk/pubsys/publishedFiles/JeGe06.pdf [Viitattu 28.3.2014].

LIITTEET

Liite 1. Virtuaalipikselitoimintoihin liittyvät apufunktiot

```
CIwUILayoutSpacer* SpacerFixed(CIwVec2 size)
{
    CIwUILayoutSpacer* spacer = new CIwUILayoutSpacer;
    spacer->SetMin(size);
    spacer->SetMax(size);
    return spacer;
}

void FetchAndApplyProperties(CIwUIElement* element, IwUIAlignmentH& alignH, IwUIAlignmentV&
alignV, CIwSVec2& border)
{
    UIManager* manager = UIManager::GetInstance();

    if (element->GetProperty("borderVP", border, true))
        border = manager->VirtualToPixels(border);
    else
        element->GetProperty("border", border, true);

    CIwVec2 sizeMin(0, 0);
    if (element->GetProperty("sizeMinVP", sizeMin, true))
    {
        sizeMin = manager->VirtualToPixels(sizeMin);
        element->SetSizeMin(sizeMin);
    }

    CIwVec2 sizeMax(0, 0);
    if (element->GetProperty("sizeMaxVP", sizeMax, true))
    {
        sizeMax = manager->VirtualToPixels(sizeMax);
        element->SetSizeMax(sizeMax);
    }

    element->GetProperty("alignH", alignH, true);
    element->GetProperty("alignV", alignV, true);
}

void AddElementToLayout(CIwUIElement* element, CIwUILayout* layout)
{
    IwUIAlignmentH alignH(IW_UI_ALIGN_CENTRE);
    IwUIAlignmentV alignV(IW_UI_ALIGN_MIDDLE);
    CIwSVec2 border(0, 0);

    FetchAndApplyProperties(element, alignH, alignV, border);

    layout->AddElement(element, alignH, alignV, border);
}

void AddSpacerToLayout(int widthVirtual, int heightVirtual, CIwUILayout* layout)
{
    layout->AddLayoutItem(SpacerFixed(UIManager::GetInstance()-
>VirtualToPixels(CIwVec2(widthVirtual, heightVirtual))));
}

void AddLayoutToLayout(CIwUILayout* layout, CIwUILayout* addInto, int16 horiBorderVirtual,
int16 vertBorderVirtual)
{
    addInto->AddLayoutItem(layout, IW_UI_ALIGN_CENTRE, IW_UI_ALIGN_MIDDLE, UI-
Manager::GetInstance()->VirtualToPixels(CIwSVec2(horiBorderVirtual, vertBorderVirtual)));
}
```

```
void ApplyVirtualMargin(CIwUILabel* label)
{
    CIwSVec2 margin;

    if (label->GetProperty("marginVP", margin, true))
        label->SetProperty("margin", UIManager::GetInstance()->VirtualToPixels(margin));
}

void ApplyVirtualMargin(CIwUIButton* button)
{
    CIwSVec2 margin;

    if (button->GetProperty("marginVP", margin, true))
        button->SetProperty("margin", UIManager::GetInstance()->VirtualToPixels(margin));
}
```

Liite 2. TTF-fontin latausprosessi

```
// Luo resurssiryhmä fonteille
CIwResGroup* fontResourceGroup = new CIwResGroup("ui_fonts");

// Avaa fonttitiedosto luettavaksi binäärimuodossa
s3eFile* file = s3eFileOpen("UI/Fonts/arial.ttf", "rb");

if (file != NULL)
{
    // Fonttidata pitää olla saatavilla niin kauan kun fontteja käytetään
    size_t fileSize = s3eFileGetSize(file);
    uint8_t* fontData = new uint8_t[fileSize];

    // Lue fonttidata muistiin
    if (s3eFileRead(fontData, fileSize, 1, file) != 0)
    {
        int virtualFontSize = 16;
        int pointSize = UIManager::GetInstance()->VirtualToPixelsRound(virtualFontSize);
        CIwGxFont* font = IwGxFontCreateTTFontFromBuffer(fontData, fileSize, pointSize);
        font->SetName("font_small");
        fontResourceGroup->AddRes(IW_GX_RESTYPE_FONT, font);

        virtualFontSize = 19;
        pointSize = UIManager::GetInstance()->VirtualToPixelsRound(virtualFontSize);
        font = IwGxFontCreateTTFontFromBuffer(fontData, fileSize, pointSize);
        font->SetName("font_medium");
        fontResourceGroup->AddRes(IW_GX_RESTYPE_FONT, font);

        virtualFontSize = 24;
        pointSize = UIManager::GetInstance()->VirtualToPixelsRound(virtualFontSize);
        font = IwGxFontCreateTTFontFromBuffer(fontData, fileSize, pointSize);
        font->SetName("font_large");
        fontResourceGroup->AddRes(IW_GX_RESTYPE_FONT, font);

        virtualFontSize = 30;
        pointSize = UIManager::GetInstance()->VirtualToPixelsRound(virtualFontSize);
        font = IwGxFontCreateTTFontFromBuffer(fontData, fileSize, pointSize);
        font->SetName("font_xlarge");
        fontResourceGroup->AddRes(IW_GX_RESTYPE_FONT, font);

        // Lisää resurssiryhmä resurssijärjestelmään
        IwGetResManager()->AddGroup(fontResourceGroup);
    }
    else
    {
        delete fontData;
        fileSize = 0;
    }

    s3eFileClose(file);
}
else
    delete fontResourceGroup;
```